Java Lecture-2

A First Simple Program

```
 /*
       This is a simple Java program.
        Call this file "Example.java".
    */
     class Example {
      // Your program begins with a call to main().
       public static void main(String args[]) {
        System.out.println("This is a simple Java program.");
       }
}
```

- In Java, all code must reside inside a class.

- By convention, the name of the main class should match the name of the file that holds the program.

- We should also make sure that the capitalization of the filename matches the class name.

- The reason for this is that Java is case-sensitive.

Compiling the Program

❖ To compile the **Example** program, execute the compiler, **javac**, specifying the name of the source file on the command line, as shown here:

```
C:\>javac Example.java
```

❖ The **javac** compiler creates a file called **Example.class** that contains the bytecode version of the program.

❖ As discussed earlier, the Java bytecode is the intermediate representation of the program that contains instructions the Java Virtual Machine will execute.

❖ Thus, the output of **javac** is not code that can be directly executed.

- To actually run the program, you must use the Java application launcher called **java**.

- To do so, pass the class name **Example** as a command-line argument, as shown here:

```
C:\>java Example
```

- When the program is run, the following output is displayed:

```
This is a simple Java program.
```

- The **public** keyword is an *access modifier.*

- **main( )** must be declared as **public**, since it must be called by code outside of its class when the program is started.

- The keyword **static** allows **main( )** to be called without having to instantiate a particular instance of the class.

- This is necessary since **main( )** is called by the Java Virtual Machine before any objects are made.

- The keyword **void** simply tells the compiler that **main( )** does not return a value.

- **main( )** is the method called when a Java application begins.

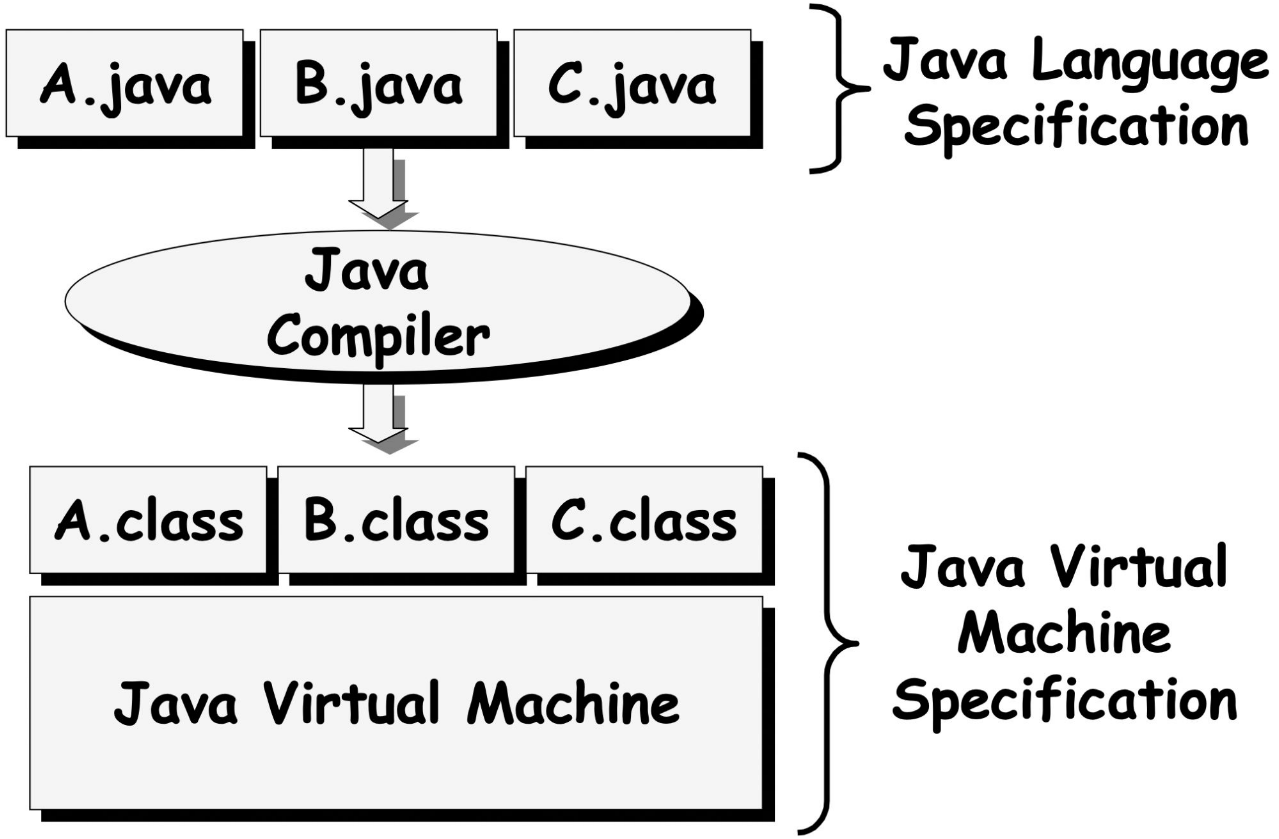
- **String args[ ]** declares a parameter named **args**, which is an array of instances of the class **String**.

- **args** receives any command-line arguments present when the program is executed.

- **println( )** displays the string which is passed to it.

- **System** is a predefined class that provides access to the system, and **out** is the output stream that is connected to the console.

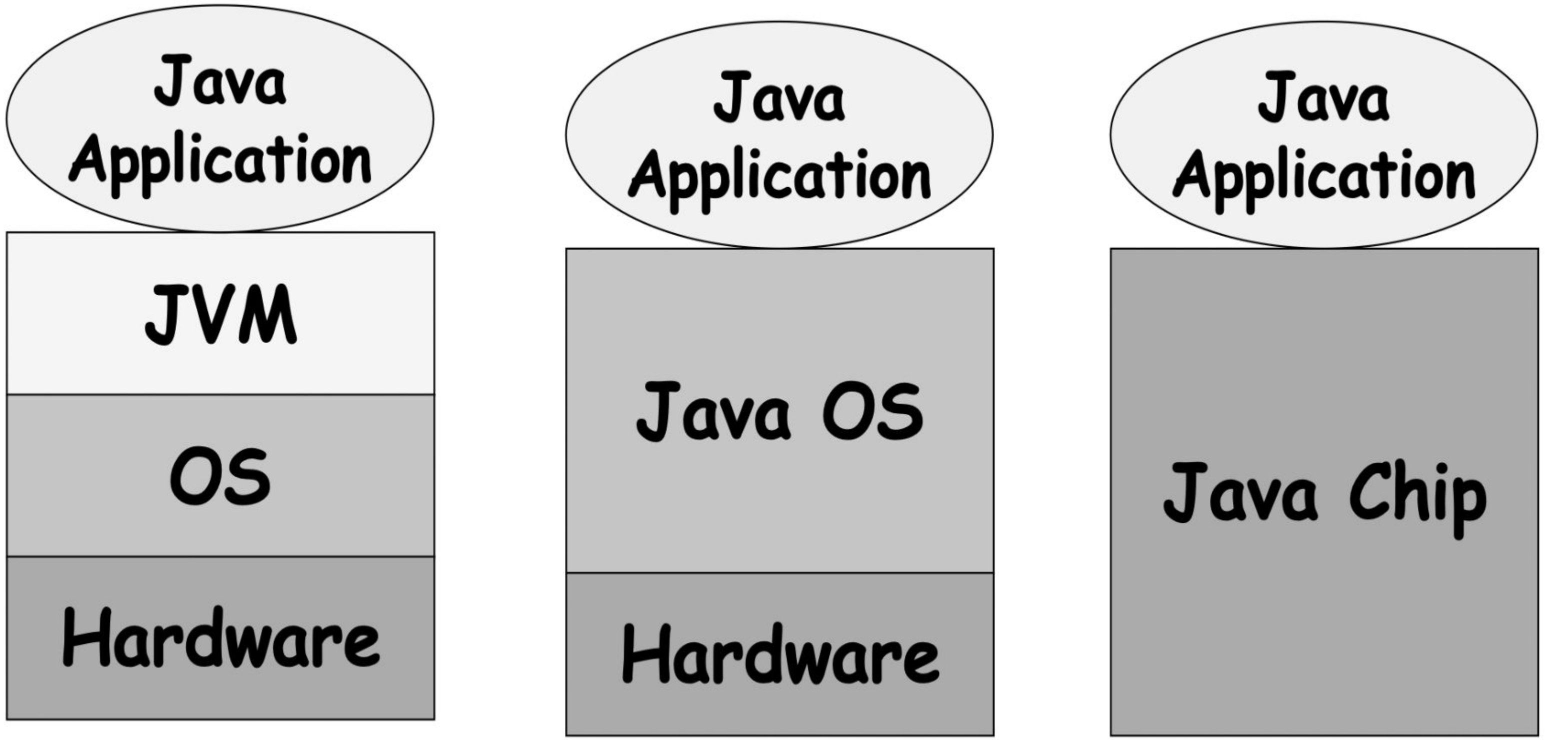
Java Virtual machine



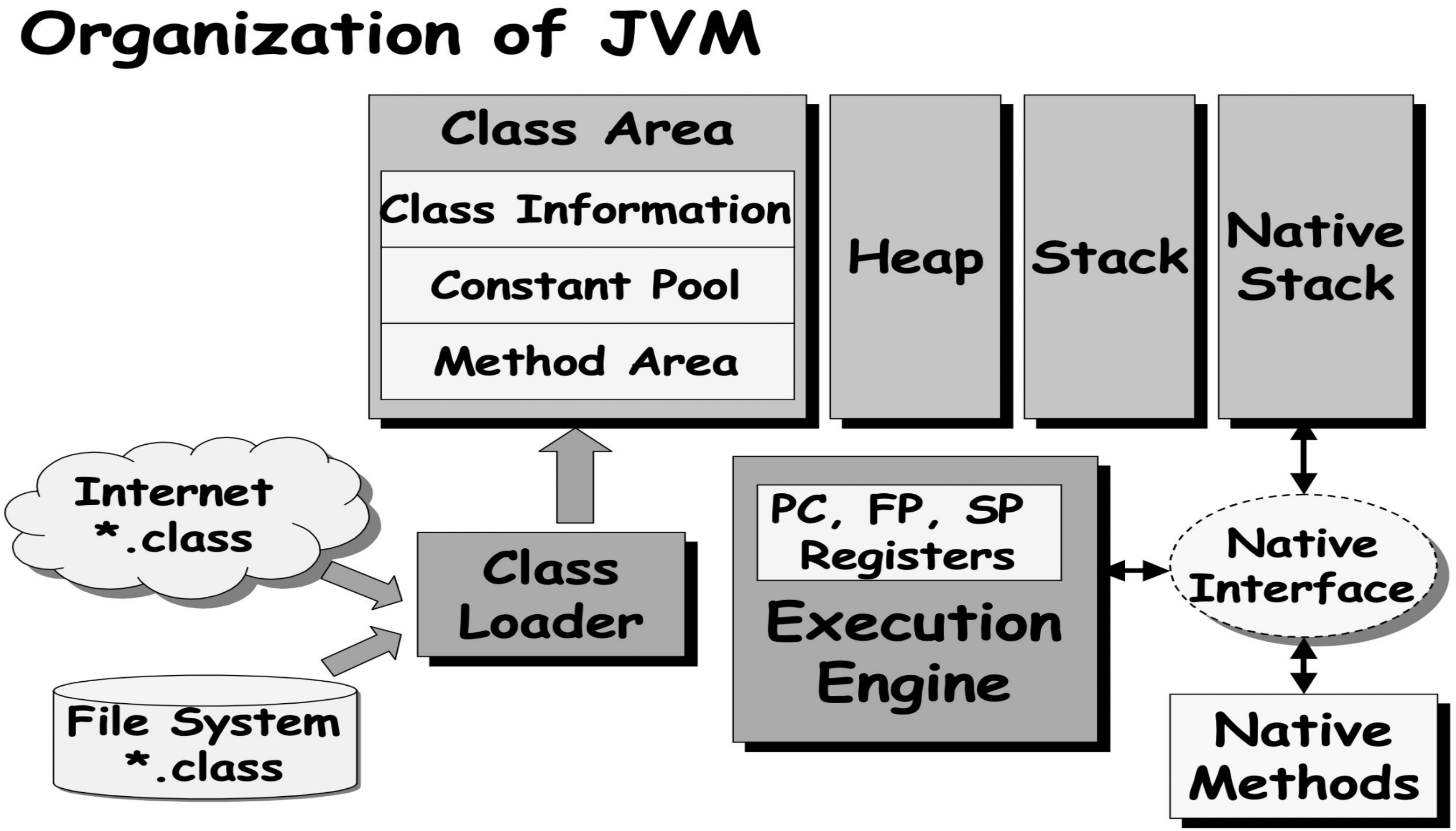
- ❖ A Java Virtual Machine (JVM) is a program that interprets Java bytecode to run as a program by providing a runtime environment that executes this process.

- ❖ One of the most significant benefits of using Java is using a JVM to run a Java program in any operating environment.

Implementations of JVM



JVM Architecture



Java Virtual Machine Architecture

Classloader: The classloader is used for loading class files. Class files are needed for the classloader to perform its three primary functions which are to link, load, and initialize.

Method Area: The JVM method area is where class structures of varying types are needed for running a java program.

Heap: All the Objects, related instance variables, and arrays are stored as common memory in the heap, where it is shared across multiple threads.

JVM Language Stacks: Java Language Stacks store local variables, and it’s partial results. Each thread has its own JVM stack, created as its thread is created. When method invocation starts, a new frame is created then deleted when method invocation is completed.

PC Registers: The PC register stores the address of the Java virtual machines currently executing instruction. In Java, each thread gets its own PC register.

Native Method Stacks: Native method stacks hold the instruction of native code written in another language instead of Java, by use of the native library.

Execution Engine: Execution Engine is a type of software used to test hardware, software, or complete systems; it does so without retaining any information about the tested product.

Native Methods Interface: The Native Method Interface is a programming framework that allows running Java code in a JVM to call libraries and native applications.

Native Methods Libraries: Native Libraries is a collection of the Native Libraries such as C languages needed by the Execution Engine.

The Java Keywords

There are 50 keywords currently defined in the Java language.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

- Java is a free-form language. This means that you do not need to follow any special indentation rules.

- Identifiers are used to name things, such as classes, variables, and methods.

- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.

- Java is case-sensitive language.

- A constant value in Java is created by using a *literal* representation of it.

Separators

In Java, there are a few characters that are used as separators. The most commonly used separator in Java is the semicolon. As you have seen, it is used to terminate statements. The separators are shown in the following table:

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.
::	Colons	Used to create a method or constructor reference. (Added by JDK 8.)

Java Is a Strongly Typed Language

- ❖ Every variable has a type, every expression has a type, and every type is strictly defined.

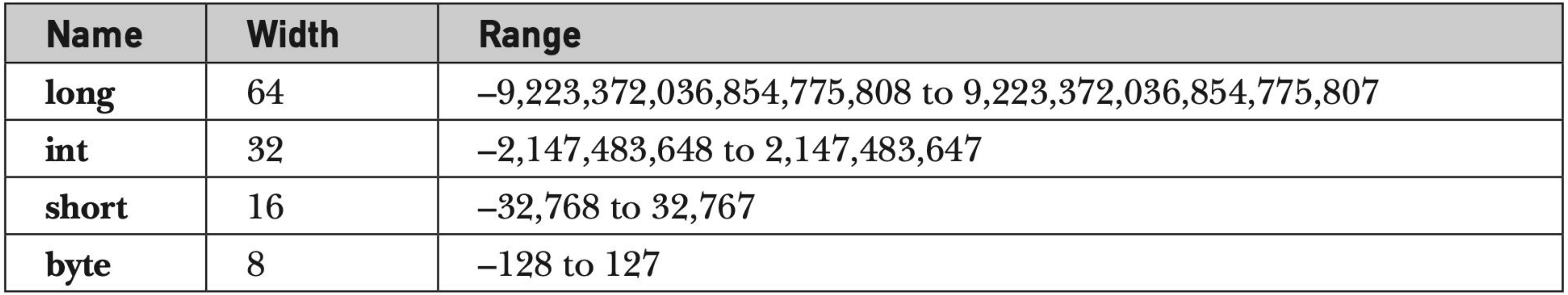
- ❖ All assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility.

- ❖ There are no automatic coercions or conversions of conflicting types

Java defines eight *primitive* types of data

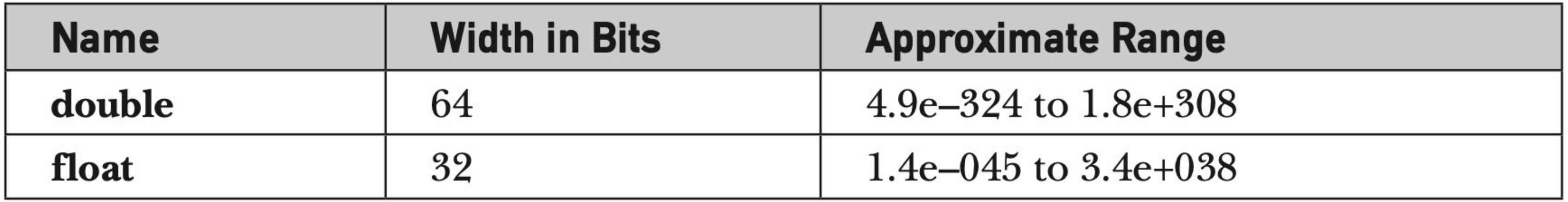
- ❖ byte,
- ❖ short,
- ❖ int,
- ❖ long,
- ❖ char,
- ❖ float,
- ❖ double, and
- ❖ boolean.

Integers



Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Floating-point Types



Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

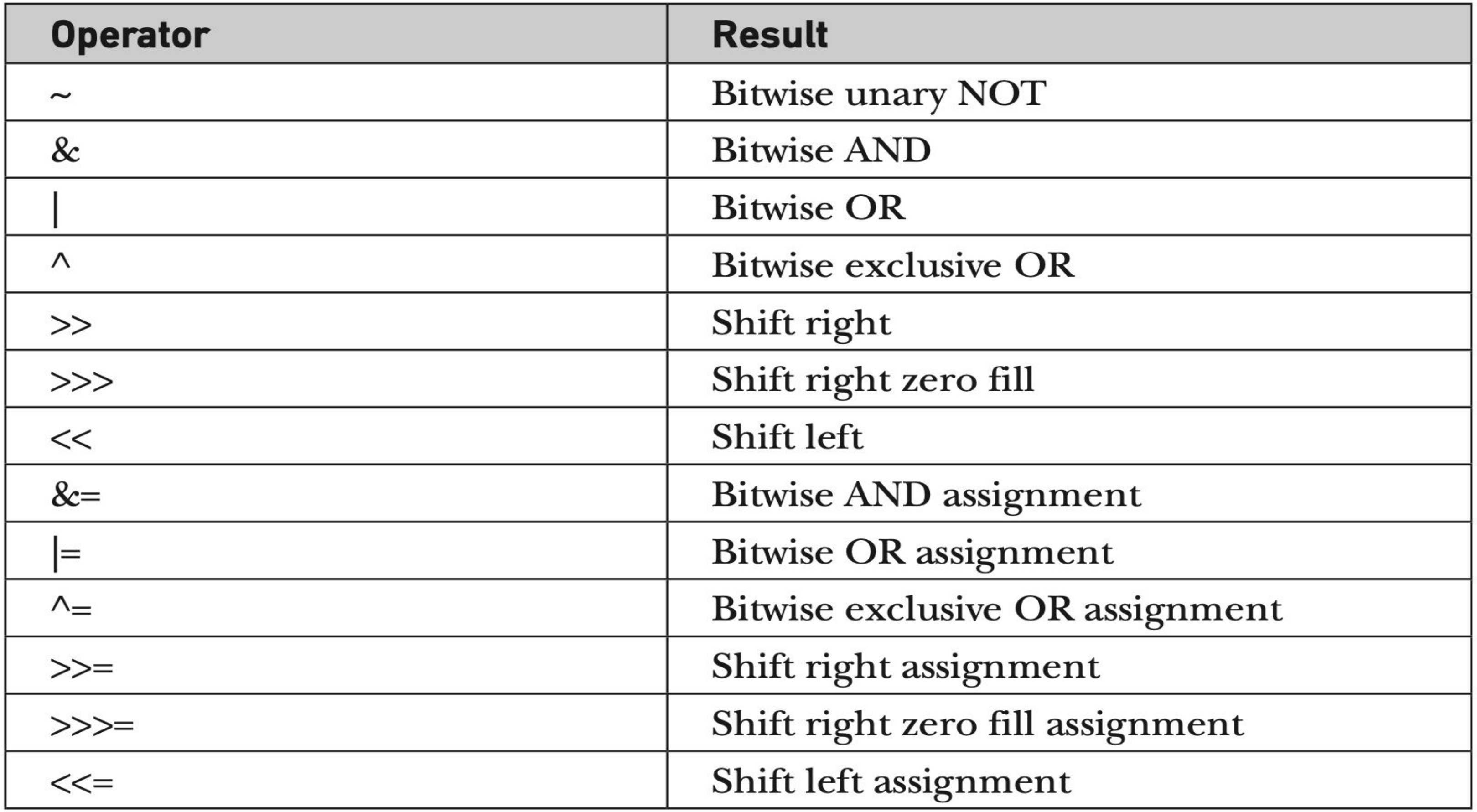
Declaring a Variable

type identifier **[=** ***value*** **][,** ***identifier*** **[=** ***value*** **] ...];**

Arithmetic Operators

Operator	Result
+	Addition (also unary plus)
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
– =	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

The Bitwise Operators



Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Relational Operators

The *relational operators* determine the relationship that one operand has to the other. Specifically, they determine equality and ordering. The relational operators are shown here:

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Boolean Logical Operators

The Boolean logical operators shown here operate only on **boolean** operands. All of the binary logical operators combine two **boolean** values to form a resultant **boolean** value.

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

The Precedence of the Java Operators

Highest						
++ (postfix)	– – (postfix)					
++ (prefix)	– – (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
->						
=	op=					
Lowest						