

Operators

And *operators* such as + and = are used to express basic computation such as addition or String concatenation or assignment.

Blocks

There are also left and right braces ({ and }) which enclose *blocks*. The body of a class is one such block.

Statements

A Block contains one or more Java statement(s), separated by semicolons. A statement is the smallest building block of Java.

Separators

Some tokens are punctuation, such as periods . and commas , and semicolons ;.

whitespace

You use *whitespace* such as spaces, tabs, and newlines, to separate tokens. For example, whitespace is required between keywords and identifiers: publicstatic is a single identifier with twelve characters, not two Java keywords.

Comments

Comments are not part of the executing code. Comments are used to document the code.

Unicode

Most Java program text consists of ASCII characters, but any Unicode character can be used as part of identifier names, in comments, and in character and string literals. Unicode escape sequences may also be used to express a Unicode character.

For example, π (which is the Greek Lowercase Letter **pi**) is a valid Java identifier.

```
double  $\pi$  = Math.PI;
```

and in a string literal

```
String pi = " $\pi$ ";
```

π may also be represented in Java as the *Unicode escape sequence* \u03C0. Thus, the following is a valid, but not very readable, declaration and assignment:

```
double \u03C0 = Math.PI;
```

The following demonstrate the use of Unicode escape sequences in other Java syntax:

```
// Declare Strings pi and quote which contain \u03C0 and \u0027
respectively:
String pi = "\u03C0";
String quote = "\u0027";
```

Note that a Unicode escape sequence functions just like any other character in the source code. E.g., \u0022 (double quote, ") needs to be quoted in a string just like ".

```
// Declare Strings doubleQuote1 and doubleQuote2 which both contain "
(double quote):
String doubleQuote1 = "\"";
String doubleQuote2 = "\\u0022"; // "\u0022" doesn't work since ""
doesn't work.
```

See Unicode escape sequences for full details.

Literals

Java Literals are syntactic representations of boolean, character, numeric, or string data. Literals provide a means of expressing specific values in your program. For example, in the following statement, an integer variable named `count` is declared and assigned an integer value. The literal `0` represents, naturally enough, the value zero.

```
int count = 0;
```

The following method call passes a String literal `"int count = 0;"` the boolean literal `true` and the special null value `null` to the method `parse()`:

```
List items = parse("int count = 0;", true, null);
```

- Boolean Literals
- Numeric Literals
 - Character Literals
 - Integer Literals
 - Floating Point Literals
- String Literals
- **null**

Blocks

Java has a concept called block that is enclosed between the `{` and `}` characters, called curly braces. A block executed as a single statement, and can be used where a single statement is accepted.

After a block is executed all local variables defined inside the block is discarded, go out of scope.

```
{  
...  
    // -- This is a block ---  
}
```

Blocks can be nested:

```
{  
...  
    {  
        // -- This is a nested block ---  
    }  
}
```

Whitespaces

Whitespace in Java is used to separate the tokens in a Java source file. Whitespace is required in some places, such as between access modifiers, type names and Identifiers, and is used to improve readability elsewhere.

Wherever whitespace is required in Java, one or more whitespace characters may be used. Wherever whitespace is optional in Java, zero or more whitespace characters may be used.

Java whitespace consists of the

- space character ' ' (0x20),
- the tab character (hex 0x09),
- the form feed character (hex 0x0c),
- the line separators characters newline (hex 0x0a) or carriage return (hex 0x0d) characters.

Line separators are special whitespace characters in that they also terminate line comments, whereas normal whitespace does not.

Other Unicode space characters, including vertical tab, are not allowed as whitespace in Java.

Required Whitespace

Below is the declaration of an **abstract** method taken from a Java class

```
public abstract Distance distanceTo(Destination dest);
```

Whitespace is required between **public** and **abstract**, between **abstract** and **Distance**, between **Distance** and **distanceTo**, and between **Destination** and **dest**.

However, the following is not legal:

```
publicabstractDistance distanceTo(Destination dest);
```

because whitespace is required between keywords and identifiers. The following is lexically valid

```
publicabstractDistance distanceTo(Destination dest);
```

but means something completely different: it declares a method which has the return type `publicabstractDistance`. It is unlikely that this type exists, so the above would result in a semantic error.

Indentation

Java ignores all whitespace in front of a statement. As this, these two code snippets are identical for the compiler:

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

```
void printMessage() {  
    System.out.println("Hello World!");  
}
```

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

```
void printMessage() {  
System.out.println("Hello World!");  
}
```