Why Java?

Object Oriented

Platform Independent

Write Once, Run Anywhere.

Simple Java Program

public class MyFirstJavaProgram {

public static void main(String []args) {

System.out.println("Hello World"); // prints Hello World

}

}

Java program processing starts from the main() method which is a mandatory part of every Java program.

Running the programs:

javac MyFirstJavaProgram.java

java MyFirstJavaProgram

The javac command reads source files that contain module, package and type declarations written in the Java programming language, and compiles them into class files that run on the Java Virtual Machine

The java command starts a Java application. It does this by starting a Java runtime environment, loading a specified class, and calling that class's main method. By default, the first argument without an option is the name of the class to be called.

**Buzz Words**

* Simple
* Secure
* Portable
* Object-oriented
* Robust
* Multithreaded
* Architecture-neutral
* Interpreted
* High performance
* Distributed
* Dynamic

**The Primitive Types**

Java defines eight *primitive* types of data: **byte**, **short**, **int**, **long**, **char**, **float**, **double**, and **boolean**. The primitive types are also commonly referred to as *simple* types, and both terms will be used in this book. These can be put in four groups:

• Integers This group includes **byte**, **short**, **int**, and **long**, which are for whole-valued signed numbers.

• Floating-point numbers This group includes **float** and **double**, which represent numbers with fractional precision.

* Characters This group includes **char**, which represents symbols in a character set, like letters and numbers.
* Boolean This group includes **boolean**, which is a special type for representing true/false values.

You can use these types as-is, or to construct arrays or your own class types. Thus, they form the basis for all other types of data that you can create.

The primitive types represent single values—not complex objects. Although Java is otherwise completely object-oriented, the primitive types are not. They are analogous to  
the simple types found in most other non–object-oriented languages. The reason for this is efficiency. Making the primitive types into objects would have degraded performance too much.

The primitive types are defined to have an explicit range and mathematical behavior. Languages such as C and C++ allow the size of an integer to vary based upon the dictates of the execution environment. However, Java is different. Because of Java’s portability requirement, all data types have a strictly defined range. For example, an **int** is always 32 bits, regardless of the particular platform. This allows programs to be written that are guaranteed to run *without porting* on any machine architecture. While strictly specifying the size of an integer may cause a small loss of performance in some environments, it is necessary in order to achieve portability.

1. The width and ranges of these integer types vary widely, as shown in this table:

page67image56291328

|  |  |  |
| --- | --- | --- |
| **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 |
| **shor t** | 16 | –32,768 to 32,767 |
| **byte** | 8 | –128 to 127 |

**Primitive Data Types**

byte - 8-bit signed

Range: -128 to 127 (-2^7 to 2^7-1)

Eg: byte a = -50

1 1 1 1 1 1 1 1

(64)(32)(16)(8)4)(2)(1)

short - 16-bit signed

Range: -32,768 to 32,767 (-2^15 to 2^15-1)

Eg: short a = 10000

int - 32-bit signed

Range: -2,147,483,648 to 2,147,483,647 (-2^31 to 2^31-1)

Eg: int height = 100000;

long - 64-bit signed

Range: (-2^63 to 2^63-1)

Eg: int height = 100000L;

float - single-precision floating point

Eg: float f1 = 234.5f

double - double-precision floating point

Range: (-2^64 to 2^64-1)

Eg: int height = 12.34;

boolean - represents one bit of information

Eg: boolean flag = true;

char - 16-bit Unicode character

Range: (-2^64 to 2^64-1)

Eg: char initial = ‘A’;

Variables and declarations

Some examples:

int a, b, c; // Declares three ints, a, b, and c.

int a = 10, b = 10; // Example of initialization

byte B = 22; // initializes a byte type variable B.

double pi = 3.14159; // declares and assigns a value of PI.

Char a = ‘a’; // the char variable a iis initialized with value ‘a’

**Two Control Statements**

**The if Statement**

**The for Loop**

**Using Blocks of Code**

if(x < y) { // begin a block

x = y;

y = 0;

} // end of block

**Lexical Issues**

**Whitespace**

Java is a free-form language. This means that you do not need to follow any special indentation rules. For instance, the **Example** program could have been written all on one line or in any other strange way you felt like typing it, as long as there was at least one whitespace character between each token that was not already delineated by an operator or separator. In Java, whitespace is a space, tab, or newline.

**Identifiers**

Some examples of valid identifiers are

page63image22615872

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AvgTemp | count | a4 | $test | this\_is\_ok |

Some examples of invvalid identifiers are

page63image22615872

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2AvgTemp | Count/1 | a-4 | 2$test | Thisisnot/ok |

**Comments**

// or /\* \*/

**Java Keywords**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | continue | for | new | switch |
| asser t | default | goto | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | impor t | public | throws |
| case | enum | instanceof | return | transient |
| catch | extends | int | shor t | try |
| char | final | inter face | static | void |
| class | finally | long | strictfp | volatile |
| const | float | native | super | while |

**String Literals**

String literals in Java are specified like they are in most other languages—by enclosing a sequence of characters between a pair of double quotes. Examples of string literals are

“Hello World” “two\nlines”  
“\”This is in quotes\”“

**Variables**

**Declaring a Variable**

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

int a, b, c;

int d = 3, e, f = 5;

byte z = 22;

double pi = 3.14159;

char x = 'x';

// declares three ints, a, b, and c.

// declares three more ints, initializing

// d and f.

// initializes z.

// declares an approximation of pi.

// the variable x has the value 'x'.

**The Scope and Lifetime of Variables**

// Demonstrate block scope.

class Scope {

public static void main(String args[]) {

int x; // known to all code within main

x = 10;

if(x == 10) { // start new scope

int y = 20; // known only to this block

// x and y both known here.

System.out.println("x and y: " + x + " " + y);

x = y \* 2;

}

// y = 100; // Error! y not known here

// x is still known here.

System.out.println("x is " + x);

}

}

**Arrays**

**One-Dimensional Arrays**

A *one-dimensional array* is, essentially, a list of like-typed variables. To create an array, you first must create an array variable of the desired type. The general form of a one-dimensional array declaration is

*type var-name*[ ];

**Multidimensional Arrays**

int twoD[][] = new int[4][5];