JAVA ASSIGNMENT

Lana Anvar

23 June 2023

Contents

DATA TYPES

Data types in Java are primarily of two types: primitive type and non-primitive type.

The Primitive Types

Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean. these are categorized further into 4 groups: int, floating-point numbers, characters, and boolean.

Integers

Java defines 4 integer types: byte, short, int, and long.

NAME	BYTES	RANGE
byte	1	-128 to 127
short	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647
long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Floating-Point Types

The floating-type data type is used to store values whose precision requires a fractional precision. The bytes and range of the floating point data types are given below :

NAME	BYTES	RANGE
float	8	4.9e-324 to 1.8e+308
double	32	1.4e-045 to 3.4e+038

Characters

Unlike C/C++, in which char is 8 bits wide, Java char is 16 bits wide to represent characters using Unicode. Unicode defines a fully international character set that can represent all of the characters found in all human languages.

Booleans

The data type boolean can have only either true or false. This is the type returned by all relational operators as well as conditional expressions.

VARIABLES

Variables are the basic unit of storage in Java. All variables have a scope to define their visibility and a lifetime. Below shown is the format of variable declaration in Java :

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

Dynamic Initialization of Variables

Java not only uses constants as initializers, but also allows dynamic initialization of variables. This is made possible by using any expression valid at the time the variable is declared.

The Scope and Lifetime of Variables

In Java, scope is defined with the help of a block, i.e, the moment a new block starts a new scope is created. The two major scopes in Java, are those defined by a class and those defined by a method. When a variable is declared within a scope, it is not accessible to the code outside that scope.

TYPE CONVERSION AND CASTING

Automatic conversion of the value of one type to another type is possible in Java only if the two types are compatible.

Automatic Conversions

The conditions for automatic conversions are as follows :

- The two types are compatible
- The destination type is larger than the source type.

For example, automatic conversion from byte to int is possible because the int type is always large enough to hold all valid byte values while the vice versa is not possible.

Casting

Conversion of a byte to an int variable is performed by *narrowing conversion*, in which the value is narrowed down to fit the target type. The general form :

(target-type) value

Here, the target type specifies the desired type to convert the specified value to.

Conversion of floating-point value to integer type is performed by *truncation*. During the conversion, the fractional component of the floating-point value is lost.

OPERATORS

In Java, operators are categorized based on their functionality into four groups in addition to other additional operators that handle certain special situations: arithmetic, bitwise, relational, and logical. Let's take a look at it in detail.

Arithmetic Operators

The operands of the arithmetic operators must be of numeric type or of char type. It cannot be used on the boolean type.

The arithmetic operators in Java are given below :

OPERATORS	**FUNCTION**
+	Addition
-	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
–	Decrement

Bitwise Operators

These operators act upon the individual bits of their operands. They are summarized in the following table:

Relational Operators

The relational operators define the relationship between operands. The outcome of these operations is a boolean value. These operators are often used along **if** statements and loop statements.

OPERATORS	FUNCTIONS
∼	Bitwise unary NOT
&	Bitwise AND
—	Bitwise OR
ˆ	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
—=	Bitwise OR assignment
ˆ=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

OPERATORS	FUNCTIONS
∼	Bitwise unary NOT
&	Bitwise AND
—	Bitwise OR
ˆ	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
—=	Bitwise OR assignment
ˆ=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Boolean Logical Operators

The boolean logical operators operate only on boolean operands. The boolean logical operators in Java are given below :

CONTROL STATEMENTS

Selection Statements

The two selection statements in Java are **if** and **switch**.

if :

if (condition) statement1; else statement2;

OPERATORS	FUNCTION
&	Logical AND
—	Logical OR
ˆ	Logical XOR (exclusive OR)
——	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
!=	OR assignment
ˆ=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

If the condition is true, then *statement1* is executed and in all other cases *statement2* is executed.

switch :

It executes the required part of the code based on the value of an expression.

```
switch (expression)
case value1:
// statement sequence
break;
case value2:
// statement sequence
break;
.
.
.
case valueN:
// statement sequence
break;
default:
// default statement sequence
```

Iteration Statements

Java's iteration statements are **for, while**and **do-while**, which executes the same code until termination condition is met.

while :

while(condition) $\{//body\ of\ loop\}$

The body of the loop will be executed as long as the conditional expression is true.

do-while :

do-while loop executes at least once, even if the condition is false, unlike while loop.

do$\{//body\ of\ loop$
$\}while(condition)$

for :

The general form of the for loop is as follows:
for(initialization; condition; iteration) $\{//body\}$

When the loop first starts, the initialization portion of the loop is executed. Loop control variable acts as a counter that controls the loop.