INTRODUCING DATA TYPES AND OPERATORS

Asmaliza Ahzan
IVERSON ASSOCIATES SDN BHD

Introducing Data Types and Operators

Why Data Types Are Important

- Java is a strongly typed language.
- Every variable, 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.
- No automatic coercions or conversions of conflicting types as in some languages.
- The Java compiler checks all expressions and parameters to ensure that the types are compatible.
- Any type mismatches are errors that must be corrected before the compiler will finish compiling the class.

Java's Primitive Types

Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean.

Group	Description
Integers	This group includes byte, short, int, and long, which are for whole-valued signed numbers.
Float-point	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.

Integers

Java defines four integer types: byte, short, int, and long. All of these are signed, positive and negative values. Java does not support unsigned, positive-only 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 points

Floating-point numbers, also known as real numbers, are used when evaluating expressions that require fractional precision. For example, calculations such as square root, or transcendentals such as sine and cosine, result in a value whose precision requires a floating point type.

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

Characters

In Java, the data type used to store characters is char. At the time of Java's creation, Unicode required 16 bits. Thus, in Java char is a 16-bit type. The range of a char is 0 to 65,536. There are no negative chars. The standard set of characters known as ASCII still ranges from 0 to 127 as always, and the extended 8-bit character set, ISO-Latin-1, ranges from 0 to 255.

Boolean

Java has a primitive type, called boolean, for logical values. It can have only one of two possible values, true or false.

Variables

The variable is the basic unit of storage in a Java program. A variable is defined by the combination of an identifier, a type, and an optional initializer. In addition, all variables have a scope, which defines their visibility, and a lifetime.

Declaring a Variable

In Java, all variables must be declared before they can be used. The basic form of a variable declaration is shown here:

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

The Scope and Lifetime of Variables

- A block defines a scope. Thus, each time you start a new block, you are creating a new scope.
- A scope determines what objects are visible to other parts of your program.
- It also determines the lifetime of those objects.

Operators

Java provides a rich operator environment. Most of its operators can be divided into the following four groups: arithmetic, bitwise, relational, and logical.

Arithmetic Operators

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

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

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

Boolean Logical Operators

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

A	B	A | B	A & B	A ^ B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

The Assignment Operator

- The assignment operator is the single equal sign, =.
- The assignment operator works in Java much as it does in any other computer language.
- It has this general form:

var = expression;

Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Using Parentheses

Parentheses raise the precedence of the operations that are inside them. This is often necessary to obtain the result you desire. For example, consider the following expression:

```
a >> b + 3
```

This expression first adds 3 to b and then shifts a right by that result. That is, this expression can be rewritten using redundant parentheses like this:

```
a >> (b + 3)
```

However, if you want to first shift a right by b positions and then add 3 to that result, you will need to parenthesize the expression like this:

```
(a >> b) + 3
```

Type Conversion in Assignments

- If the two types are compatible, then Java will perform the conversion automatically. For example, it is always possible to assign an int value to a long variable.
- However, not all types are compatible, and thus, not all type conversions are implicitly allowed. For instance, there is no automatic conversion defined from double to byte.
- Fortunately, it is still possible to obtain a conversion between incompatible types.

- To do so, you must use a cast, which performs an explicit conversion between incompatible types.

Java's Automatic Conversions

When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

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

When these two conditions are met, a widening conversion takes place. For example, the int type is always large enough to hold all valid byte values, so no explicit cast statement is required.

For widening conversions, the numeric types, including integer and floating-point types, are compatible with each other. However, there are no automatic conversions from the numeric types to char or boolean. Also, char and boolean are not compatible with each other.

Casting Incompatible Types

To create a conversion between two incompatible types, you must use a cast. A cast is simply an explicit type conversion. It has this general form:

(*target-type*) *value*

Expressions

An expression is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language, that evaluates to a single value.

```
int cadence = 0;
anArray[0] = 100;
System.out.println("Element 1 at index 0: " + anArray[0]);

int result = 1 + 2; // result is now 3
if (value1 == value2)
    System.out.println("value1 == value2");
```