Java Input/Output

Java uses streams to perform input and output. A stream can be defined as a sequence of data. All streams represent an input source and an output destination. Java uses the InputStream to read data from a source and the OutputStream is used for writing data to a destination.

The JDK has two sets of I/O packages:

1. the Standard I/O (in package java.io), introduced since JDK 1.0 for stream-based I/O, and
2. the New I/O (in packages java.nio), introduced in JDK 1.4, for more efficient buffer-based I/O.

Console Output

```
System.out.print("Hello ");
System.out.println("world");
```

Console Input

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
String text = in.readLine();
```

GUI Input

GUI input and output is performed using the JOptionPane class of package javax.swing

```
String name = JOptionPane.showInputDialog(frame, "What's your name?");
```

GUI Output

```
JOptionPane.showMessageDialog(null,”Welcome to Java Programming”);
```

Example:

```
import javax.swing.JOptionPane;
public class JOptionPaneTest1 {
   public static void main(String[] args) {
      String ans;
      ans = JOptionPane.showInputDialog(null, "Speed in miles per hour?");
      double mph = Double.parseDouble(ans);
      double kph = 1.621 * mph;
      JOptionPane.showMessageDialog(null, "KPH = " + kph);
      System.exit(0);
   }
}
```

File Output

```
PrintWriter out = new PrintWriter(new FileWriter("K:\\location\\outputfile.txt")));
out.print("Hello ");
out.println("world");
out.close();
```

File Input

```
BufferedReader in = new BufferedReader(new FileReader("K:\\location\\inputfile.txt"));
String text = in.readLine();
in.close();
```

Converting input data

```
String text  = in.readLine();
int x = Integer.parseInt(text);
double y = Double.parseDouble(text);
```

Reading until EOF

```
while (in.ready()) {
  text = in.readLine();
  System.out.println(text);
}
```

Variables:

Variables are named memory locations that store data to be used in a program. Variables maintain the state of your application, and enable the user to manage data. The values stored in variables may change during program execution.

Identifier:

These are variable names. Variable name is a name given to memory cells location of a computer where data is stored. You can use any combination of those characters as long as the name doesn't begin with a number. Identifier is the name of a variable that is made up from combination of alphabets, digits and underscore.

*** Rules for varibales:**

- Variable names may contain only letters, numbers, underscores.
- Should not begin with a letter. First character should be letter or alphabet.
- Keywords are not allowed to use as a variable name.
- White space is not allowed.
- Java is case sensitive i.e. UPPER and lower case are significant.
- Only underscore, special symbol is allowed between two characters.

Keywords

Keywords are the system defined identifiers. All keywords have fixed meanings that do not change. White spaces are not allowed in keywords. Keyword may not be used as an identifier. Common java keywords include:

Keyword	What It Does
abstract	Indicates that the details of a class, a method, or an interface are given elsewhere in the code.
assert	Tests the truth of a condition that the programmer believes is true.
boolean	Indicates that a value is either true or false.
break	Jumps out of a loop or switch.
byte	Indicates that a value is an 8-bit whole number.
case	Introduces one of several possible paths of execution in aswitch statement.
catch	Introduces statements that are executed when something interrupts the flow of execution in a try clause.
char	Indicates that a value is a character (a single letter, digit, punctuation symbol, and so on) stored in 16 bits of memory.
class	Introduces a class – a blueprint for an object.
const	You can't use this word in a Java program. The word has no meaning. Because it's a keyword, you can't create a constvariable.
continue	Forces the abrupt end of the current loop iteration and begins another iteration.
default	Introduces a path of execution to take when no case is a match in a switch statement.
do	Causes the computer to repeat some statements over and over again (for example, as long as the computer keeps getting unacceptable results).
double	Indicates that a value is a 64-bit number with one or more digits after the decimal point.
else	Introduces statements that are executed when the condition in an if statement isn't true.
enum	Creates a newly defined *type* — a group of values that a variable can have.
extends	Creates a subclass — a class that reuses functionality from a previously defined class.
final	Indicates that a variable's value cannot be changed, that a class's functionality cannot be extended, or that a method cannot be overridden.
finally	Introduces the last will and testament of the statements in a try clause.
float	Indicates that a value is a 32-bit number with one or more digits after the decimal point.
for	Gets the computer to repeat some statements over and over again (for example, a certain number of times).

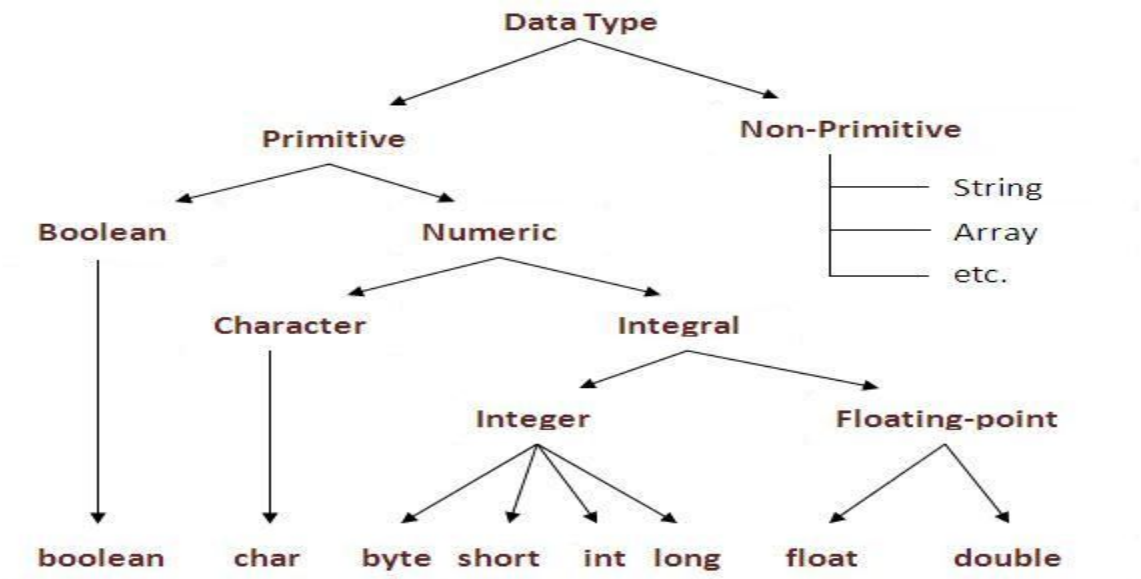
goto	You can't use this word in a Java program. The word has no meaning. Because it's a keyword, you can't create a goto variable.
if	Tests to see whether a condition is true. If it's true, the computer executes certain statements; otherwise, the computer executes other statements.
implements	Reuses the functionality from a previously defined interface.
import	Enables the programmer to abbreviate the names of classes defined in a package.
instanceof	Tests to see whether a certain object comes from a certain class.
int	Indicates that a value is a 32-bit whole number.
interface	Introduces an interface, which is like a class, but less specific. (Interfaces are used in place of the confusing multiple-inheritance feature that's in C++.)
long	Indicates that a value is a 64-bit whole number.
native	Enables the programmer to use code that was written in another language (one of those awful languages other than Java).
new	Creates an object from an existing class.
package	Puts the code into a package — a collection of logically related definitions.
private	Indicates that a variable or method can be used only within a certain class.
protected	Indicates that a variable or method can be used in subclasses from another package.
public	Indicates that a variable, class, or method can be used by any other Java code.
return	Ends execution of a method and possibly returns a value to the calling code.
short	Indicates that a value is a 16-bit whole number.
static	Indicates that a variable or method belongs to a class, rather than to any object created from the class.
strictfp	Limits the computer's ability to represent extra large or extra small numbers when the computer does intermediate calculations on float and doublevalues.
super	Refers to the superclass of the code in which the word*super* appears.
switch	Tells the computer to follow one of many possible paths of execution (one of many possible cases), depending on the value of an expression.
synchronized	Keeps two threads from interfering with one another.

this	A self-reference — refers to the object in which the word *this* appears.
throw	Creates a new exception object and indicates that an exceptional situation (usually something unwanted) has occurred.
throws	Indicates that a method or constructor may pass the buck when an exception is thrown.
transient	Indicates that, if and when an object is serialized, a variable's value doesn't need to be stored.
try	Introduces statements that are watched (during runtime) for things that can go wrong.
void	Indicates that a method doesn't return a value.
volatile	Imposes strict rules on the use of a variable by more than one thread at a time.
while	Repeats some statements over and over again (as long as a condition is still true).

Data Types in Java

Data types dictate the kind of data that a variable can store. Data type can be defined as the type of data for a variable or constant store. When we use a variable in a program then we have to mention the type of data.

In java, there are two types of data types : primitive data types and non-primitive data types



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Escape Sequence Characters (Backslash Character Constants) in C:

Java language supports few special escape sequences for String and char that are used to do special tasks. These are also called as 'Backslash characters'.

Notation	Character represented
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\‘	Double quote
\’	Single quote
\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

Variable Initialization

In Java, variables aren't necessarily assigned an initial value when they're declared, but all variables ***must*** be assigned a value before the variable's value is ***used*** in an assignment statement.

```
int foo;
int bar;
foo = 3;
// We're now initializing BOTH variables explicitly.
bar = 7;
foo = foo + bar;
```

Use of variables before they are initialized results in a compiler error:

The String Type

A String represents a sequence of zero or more Unicode characters. The symbol String starts with a capital "S," whereas the names of primitive types are expressed in all lowercase: int, float, boolean, etc.
E.g. String name = "Steve";
This capitalization difference is deliberate and mandatory—string (lowercase) won't work as a type.

String concatenation

The plus sign (+) operator is normally used for arithmetic addition, but when used in conjunction with Strings. Any number of String values can be concatenated with the + operator, as the following code snippet illustrates:

```
String x = "foo";
String y = "bar";
String z = x + y + "!"; // z assumes the value "foobar!" (x and y's values are// unaffected)
```

Constants in Java

A constant is an entity that doesn't change during the execution of a program.

Statements

A statement is basically any declaration, function call, assignment, or condition. Statements are the building blocks of any program. Statements in Java must be terminated with a semi colon. E.g.

```
int a;

a=10;
```

Comments

Java supports three different comment styles: single line (end-of-line), Multi-line and Java documentation comments.

Single line (End-of-Line Comments)

We use a double slash (//) to note the beginning of a comment that automatically ends when the end of the line is reached. E.g.

```
x = y + z; // text of comment continues through to the end of the line
```

Multi-line Comments

These comments begin with a forward slash followed by an asterisk (/*) and end with an asterisk followed by a forward slash (*/). Everything enclosed between these delimiters is treated as a comment and is therefore ignored by the Java compiler, no matter how many lines the comment spans.

Java Documentation Comments

The third and final type of Java comment, **Java documentation comments** (aka **Javadoc comments**), can be parsed from source code files by a special javadoccommand-line utility program (which comes standard with the Java SDK) and used to automatically generate HTML documentation for an application.

Operators and Expressions

An expression is a statement that has a value. Expressions are basically any math or logical operation(s). They *consist of* a sequence of operators and operands that specifies a computation. E.gx = 12+42; that assigns 42 to variable x;

A **simple expression** in Java is either

- A constant: 7, false
- A char(acter) literal enclosed in single quotes: 'A', '3'
- A String literal enclosed in double quotes: "foo", "Java"
- The name of any properly declared variables: myString, x
- Any ***two*** of the preceding types of expression that are combined with one of the Java **binary operators** (discussed in detail later in this chapter): x + 2
- Any ***one*** of the preceding types of expression that is modified by one of the Java **unary operators** (discussed in detail later in this chapter): i++
- Any of the preceding types of expression enclosed in parentheses: (x + 2) plus a few more types of expression having to do with objects that you'll learn about later in the book.

Expressions of arbitrary complexity can be assembled from the various different simple expression types by nesting parentheses, for example: ((((4/x) + y) * 7) + z).

Operators

Operations are used to perform arithmetic and logical operations when writing expressions. Common operators include:

Assignment (=) The assignment operator assigns a value to a variable. a = 5;

Arithmetic operators (+, -, *, /, %)

+	addition
-	subtraction
*	multiplication
/	division
%	modulo

Compound assignment (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

When we want to modify the value of a variable by performing an operation on the value currently stored in that variable we can use compound assignment operators which include:

expression	is equivalent to
value += increase;	value = value + increase;
a -= 5;	a = a - 5;
a /= b;	a = a / b;
price *= units + 1;	price = price * (units + 1);

// compound assignment operators

Increase and decrease (++, --)

The increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to `+=1` and to `-=1`, respectively. Thus:

c++;c+=1;c=c+1; are all equivalent in its functionality.

Relational and equality operators (==, !=, >, <, >=, <=)

These operators are used for comparison between two expressions. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result. We may want to compare two expressions, for example, to know if they are equal or if one is greater than the other is. The relational and equality operators that can be used include:

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

Logical operators (!, &&, ||)

The Operator! Is used to perform the Boolean operation NOT, it has only one operand, located at its right, and the only thing that it does is to inverse the value of it, producing false if its operand is true and true if its operand is false.

The logical operators && and || are used when evaluating two expressions to obtain a single relational result. The operator && corresponds with Boolean logical operation AND. This operation results true if both its two operands are true, and false otherwise. The following panel shows the result of operator && evaluating the expression a && b:

&& OPERATOR

A	b	a && b
True	true	true
True	false	false
False	true	false
False	false	false

The operator || corresponds with Boolean logical operation OR. This operation results true if either one of its two operands is true, thus being false only when both operands are false themselves. Here are the possible results ofa || b:

|| OPERATOR

A	b	a || b
True	true	true
True	false	true
False	true	true
False	false	false

Conditional operator (?)

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false. Its format is:
condition ?result1 : result2

Precedence of operators

Precedence of operators determines which operand is evaluated first and which one is evaluated later. For example, in this expression: a = 5 + 7 % 2 we may doubt if it really means:

```
1 a = 5 + (7 % 2)    // with a result of 6, or
2 a = (5 + 7) % 2    // with a result of 0
```

From greatest to lowest priority, the priority order is as follows:

Level	Operator	Description	Grouping
1	::	Scope	Left-to-right
2	() [] . -> ++ -- dynamic_caststatic_castreinterpret_castconst_casttypeid	Postfix	Left-to-right
3	++ -- ~ ! sizeof new delete	unary (prefix)	Right-to-left
	* &	indirection and reference (pointers)	
	+ -	unary sign operator	
4	(type)	type casting	Right-to-left
5	.* ->*	pointer-to-member	Left-to-right
6	* / %	Multiplicative	Left-to-right
7	+ -	Additive	Left-to-right
8	<<>>	Shift	Left-to-right
9	<><= >=	Relational	Left-to-right
10	== !=	Equality	Left-to-right
11	&	bitwise AND	Left-to-right
12	^	bitwise XOR	Left-to-right
13	|	bitwise OR	Left-to-right
14	&&	logical AND	Left-to-right
15	||	logical OR	Left-to-right
16	?:	Conditional	Right-to-left
17	= *= /= %= += -= >>= <<= &= ^= |=	Assignment	Right-to-left
18	,	Comma	Left-to-right

Precedence of Operators

The precedence of C operators dictates the order of calculation within an expression. The precedence of the operators introduced here is summarised in the table below. The highest precedence operators are given first.

Operators								Associativity
( )	->	.						left to right
!	~	+	-	++	--	&	*	right to left
*	/	%						left to right
+	-							left to right
<	<=	>	>=					left to right
==	!=							left to right
&								left to right
|								left to right
&&								left to right
||								right to left
=	*=	/=	%=	+=	-=			right to left

Where the same operator appears twice (for example *****) the first one is the unary version.

At the heart of most programs are pieces of code that are executed many times, with some choices between the paths through the code. C offers counted and conditional loops. Selection is offered by an **if statement** on a **multi-way switch**.

Summary:

Operator Name	Operators
Assignment	=
Arithmetic	+, -, *, /, %
Logical	&&, ||, !
Relational	<, >, <=, >=, ==, !=
Shorthand	+=, -=, *=, /=, %=
Unary	++, --
Conditional	()?:;
Bitwise	&, |, ^, <<, >>, ~

Types of expressions

Arithmetic expressions

A **unary** expression consists of either a unary operator prepended to an operand, or the sizeof keyword followed by an expression. A **binary** expression consists of two operands joined by a binary operator. A **ternary** expression consists of three operands joined by the conditional-expression operator.

Arithmetic expressions

The easiest example of an expression is in the assignment statement. An expression is evaluated, and the result is saved in a variable. A simple example might look like y = (m * x) + c

This assignment will save the value of the expression in variable y.

Short hand assignments

Shorthand	Equivalent
X=a*b++	X=a*b b=b+1
X=--i*(a+b)	I=i-1 X=i*(a+b)
i+=10	I=i+10
i-=10	I=i-10
I*=10	I=i*10
i/=10	I=i/10

Logical expressions

Logical connectors allow several comparisons to be combined into a single test. They use And, Or and Not operators.

Symbol	Meaning
&&	And
||	Or
!	Not

They are frequently used to combine relational operators, for example

x< 20 && x >= 10
They evaluate their left hand operand, and then only evaluate the right hand one if this is required. Not operates on a single logical value, its effect is to reverse its state. Here is an example of its use. if (! acceptable)

printf("Not Acceptable !!\n");

Comparison Expressions

Comparison takes two numbers and produces a logical result. Comparisons are usually found controlling if statements or loops. Example

C notation	Meaning
==	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to

Note that == is used in comparisons and = is used in assignments. Comparison operators are used in expressions like the ones below.

x == y

i> 10

a + b != c