# **Language Fundamentals**

* **Identifiers:**

A name in java program is called an identifier which can be used for identification purpose.

It can be method name, variable name, class name or label name.

**E.G:**

**Class Test {**

**Public static void main(String[] args)**

**{**

**Int x=10;**

**}**

**}**

1 Test – Name of class

2 main – Name of method

3 String – Predefined java class name

4 args – Name of array

5 x – Name of variable

**Rules for defining java identifiers:**

1. The only allowed characters in java identifiers are

* A to Z
* a to z
* 0 to 9
* $
* \_

Any other characters except these we will get compile time error

1. Identifiers cannot start with a digit.

Valid: total123

Invalid: 123total

1. Java language is treated as case sensitive programming language so Identifiers are case sensitive.

Number= 10 , number = 10 , NUMBER =10 all three are different variables.

1. There is no length limit for java identifiers but its not recommended to take too lengthy identifiers.
2. Reserved words cannot be used as identifiers.

Valid : int x=10;

Invalid : int if =10;

Valid: int String = 888; … (Predefined class name)

Valid: int Runnable = 999; … (Predefined Interface name)

1. All Predefined class names and interface names can be used as identifiers .Even though it is valid it is not recommended at all as it reduces readability and creates confusion.

**Examples**:

1. total\_number
2. total#
3. 123total
4. total123
5. ca$h
6. \_$\_$\_$\_$\_$\_
7. all@hands
8. Java2Share
9. Integer
10. Int
11. int

* **Reserved Words**

In java some words are reserved to represent some meaning or functionality. Such type of words are called reserved words

Reserved Words(53)

Keywords(50) Reserved Literals(3)

* True
* False
* Null

Used Keywords(48) Unused Keywords(2)

* goto
* const

Keywords for Data types (8):

byte short

int long

float double

boolean char

Keywords for flow control (11):

if else

switch case

default while

do for

break continue

return

Keywords for modifiers (11):

Public private

Protected static

final abstract

synchronized native

strictfp transient

volatile

Keywords for exception handling (6):

try catch

finally throw

throws assert

Class related keywords (6):

class interface

extends implements

package import

Object related keywords (4):

new instanceof

super this

Void return type keyword (1):

void

In java return type is mandatory. If a method won’t return anything we have to declare that method with void return type.

But in C language return type is optional and the default return type is int.

Unused Keywords (2):

goto

Usage of goto created several problems in old languages and hence usage of goto is banned in java.

Const

Use final instead of const.

NOTE: goto and const are unused keywords and if we try to use these we will get compile time error.

Reserved Literals (3):

true false null

true and false are values of boolean data type.

Null is default value for object reference.

Enum keyword (1):

enum

We can use enum to define a group of named constants.

**CONCLUSION:**

--> All 53 reserved words in java contain only lower-case alphabet symbols

--> In java we only have ‘new’ keyword and there is no ‘delete’ keyword because destruction of useless objects is the responsibility of garbage collector.

--> The following are new keywords in java.

* strictfp- 1.2 version
* assert- 1.4 version
* enum- 1.5 version

**strictfp** but not strictFp

**instanceof** but not instanceOf

**synchronized** but not synchronize

**extends** but not extend

**implements** but not implement

**import** but not imports

**const** but not constant

* **Data Types**

In java every variable and every expression has some type. Each and every data type is clearly defined. Every assignment should be checked by compiler for type compatibility .

Because of above reason we can conclude java language is **strongly typed** programming language .

Java is not considered as pure object oriented programming language as several OOP features are not satisfied by java (like operator overloading n multiple inheritance etc.). Moreover we are depending on primitive data types which are not objects.

Primitive DT

Numeric Non-numeric

***char*** ***boolean***

Integral DT Non-integral

***float*** ***double***

***byte*** ***short*** ***int*** ***long***

Except Boolean and char remaining data types are considered as signed data types because we can represent both the positive and negative numbers.

1. **byte:**

Size: 1 byte / 8 bits

Max Value: +127

Min Value: -128

Range: -128 to +127

The MSB acts as sign bit . 0 means positive and 1 means negative number.

Positive nos. will be represented directly in the memory whereas negative numbers will be

represented in 2’s complement form

E.g :

* byte b=10;
* byte b=127;
* byte b=128;

Compilation error: Possible loss of precision. Found int ; required byte

* byte b=10.5;

Compilation error: Possible loss of precision. Found double; required byte

* byte b=true;

Compilation error: Incompatible types. Found Boolean; required byte

* byte b=”durga”

Compilation error: Incompatible types. Found java.lang.String; required byte

Byte is the best choice if we want to handle data in terms of streams either from the file or network. File supported form or n/w supported form is byte.

1. **short**

Size: 2 bytes / 16 bits

Range: -2^15 to +2^15 -1 -32768 to +32767

This is the most rarely used data type in java.

E.g:

* Short x = 32768;

Unresolved Compilation Problem:

Type mismatch: cannot convert from int to short

* Short x = true;

Unresolved Compilation Problem:

Type mismatch: cannot convert from boolean to short

Short Data type is best suitable for 16 bit processors like 8085 but these processors are completely outdated and hence corresponding short data type is also outdated data type.

1. **int**

Size: 4 bytes / 32 bits -2147483648 to 2147483627

Range: 2^31 to +2^31 -1

It is the most commonly used data type in java.

* int x = 2147483648;

The literal 2147483648 of type int is out of range

* int x = 2147483648L;
* Type mismatch: cannot convert from long to int

1. **Long**

Size: 8 bytes / 64 bits

Range: -2^63 to +2^63 -1

**Sometimes** int may not be enough to hold big values then we should go for long type.

E.g:

* Amount of distance travelled by light in 1000 days.
* The number of characters present in a big file may exceed int range .Hence the return type of length is long and not int.

long f = f.length()

All the above data types (byte, short, int ,long) are meant for representing integral values. If we want to represent floating values then we should go for floating point data types.

1. **Floating point data types**

|  |  |
| --- | --- |
| Float | Double |
| Used when 5-6 decimal places of accuracy then go for float | Used when 14-15 decimal places of accuracy then go for float |
| Follows single precision format | Follows double precision format |
| Size: 4 bytes | Size: 8 bytes |
| Range: -3.4e38 to 3.4e38 | -1.7e308 to 1.7e308 |

1. **Boolean**

Size: Not applicable (**Virtual machine dependent**)

Range: Not applicable (Allowed values are true / false)

E.g:

* boolean b=0;

Type mismatch: cannot convert from int to boolean

* boolean b=True;

True cannot be resolved to a variable

* boolean b=”True”;

Type mismatch: cannot convert from String to boolean

* **public** **static** **void** main(String[] args) {

**while**(1)

System.***out***.print("jie");

}

* **public** **static** **void** main(String[] args) {

**int** x=0;

**if**(x)

System.***out***.print(**false**);

**else**

System.***out***.print(**true**);

}

Type mismatch: cannot convert from int to boolean

1. **char**

Size: 2 bytes / 16 bits

Range: 0 to 65535

Old languages (C or C++) are ASCII code based and the number of different ASCII code characters are <=256. To represent these 256 characters 8 bits are enough. Hence the size of char in old languages is 1 byte.

But java is Unicode based and the number of different Unicode characters are >256 and <=65535. To represent these many characters 8 bits may not be enough. So we should go for 16 bits. Hence the size of char in java is 2 bytes.

**Summary of data types**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Types | Size | Range | Wrapper class | Default Value |
| byte | 1 bytes | -128 to 127 | Byte | 0 |
| short | 2 bytes | -32768 to 32767 | Short | 0 |
| int | 4 bytes | -2147483648 to 2147483647 | Integer | 0 |
| long | 8 bytes | -2^63 to 2^63 -1 | Long | 0 |
| float | 4 bytes | -3.4e38 to 3.3e38 | Float | 0.0 |
| double | 8 bytes | -1.7e308 to 1.7e308 | Double | 0.0 |
| boolean | NA | NA[true/false] | Boolean | False |
| char | 2 bytes | 0 to 65535 | Character | 0[represents space character] |

**NOTE:**

**null** is the defau**lt** value for **object reference** which cannot be applied to primitive data types.

If we use it for primitives we will get compile time error

* **Literal**

A constant value which can be assigned to a variable is called a literal.

E.g: int x = 20;

20 is the constant value or literal

***Integral literals:***

For integral data types(byte, short, int, long) we can specify literal value in the following ways.

1. Decimal : Base 10

Allowed digits are 0 to 9. E.g: int x=0;

1. Octal : Base 8

Allowed digits are 0 to 7. Literal value should prefixed with 0. E.g: int x = 010;

1. Hexadecimal : Base 16

Allowed digits are 0 to 9 and ‘a’ to ‘f’ or ‘A’ to ‘F’. We can use both lower case and upper case characters for extra digits (‘a’ to ‘f’ and ‘A’ to ‘F’).

This one of very few areas where java is not case sensitive.

The literal value should be prefixed with 0x or 0X.

E.g: int x = 0X10;

These are the only possible ways to specify literal values for integral data types.

Examples:

* **int** x=078; …invalid

The literal 078 of type int is out of range

* int x=0Xface; …valid
* int x=0xbeef; …valid
* int x=0Xbeer; …invalid

Syntax error on token "r", delete this token

* **public** **static** **void** main(String[] args)

{

**int** x=10;

**int** y=010;

**int** z=0x10;

System.***out***.print(x+"..."+y+"..."+z);

}

**Output** : 10…8…16

The programmer has the choice to take values in decimal octal or hexadecimal form. But JVM only prints value in decimal.

By default every integral literal is of int type but we can specify explicitly as long type by suffixed with ‘l’ or ‘L’.

E.g : int x=10; ..valid

long x=10L; ..valid

int x=10L; ..invalid

long x=10 ..valid

There is no direct way to specify byte and short literals explicitly but indirectly we can specify.

Whenever we are assigning integral literal to byte variable and its value is within the range of byte then compiler treats automatically as byte literal.

Similarly short literal also.

***Floating point Literals:***

By default every floating point literal is of double type and hence we cannot directly assign to the float variable but we can specify floating point literal as float type by suffixed with ‘f’ or ‘F’.

E.g:

* float f=123.456; ..invalid
* float f=123.456f; ..valid
* double d=123.456; ..valid

We can specify explicitly floating point literal as double type by suffixing with ‘d’ or ‘D’ .ofcourse this convention is not required.

E.g:

* double d=123.456d; ..valid
* float f=123.456d; ..invalid

We can specify floating point literals only in decimal form and we cannot specify it in octal and hexadecimal forms.

If a floating point literal is prefixed with 0 then too it is considered as decimal.

E.g:

* double d=123.456; ..valid
* double d=0123.456; ..valid (considered as decimal)
* double d=0X123.456; ..invalid
* double d=0786; ..**Invalid**

If a value with no decimal point is assigned to a double variable by default it is considered as integer. Integer suffixed with 0 represents octal value. In octal representation digit ‘8’ is not allowed. Therefore, compile time error is encountered.

The literal 0786 of type int is out of range

* double d = oXface; ..valid

Considered as integer and therefore hexadecimal.

Prints : 64206.0 (o/p is always printed in decimal by JVM)

* double = 0786.0; ..valid

floating point literal hence considered as decimal.

* double = 0XFace; ..invalid

Floating point literal and therefore considered as decimal;

We can assign integral literals directly to floating point variables and that integral literal can be specified in either decimal octal or hexadecimal forms.

We cannot assign floating point literals to integral types.

* double d = 10; ..valid
* int d = 10.0; ..invalid

Type mismatch: cannot convert from double to int

* double d = 1.2e3 ..valid

1.2e3 = 1.2 \* 10^3 = 1.2 \* 1000 = 1200.0

* float f = 1.2e3; ..invalid

Type mismatch: cannot convert from double to float

* float f = 1.2e3f; ..valid

We can specify floating point literal even in exponential form (scientific notation)

***Boolean Literals:***

The only allowed values for Boolean data type are true and false

* boolean b = true; ..valid;
* boolean b = True ..invalid
* boolean b = 0; ..invalid
* boolean b = “true”; ..invalid

***Character Literals:***

We can specify char literal as single character within single quotes.

* char ch = ‘a’; ..valid;
* char ch = a ..invalid

Unresolved compilation problem:

a cannot be resolved to a variable

* char ch = “a”; ..invalid

Type mismatch: cannot convert from String to char

* char ch = ‘ab’; ..invalid
* Unresolved compilation problem:

Invalid character constant

We can specify char literal as integral literal which represents unicode value of the character and that integral literal can be specified either in decimal octal or hexadecimal forms but allowed range is 0 to 65535

* char ch = 97; ..valid
* char ch = 0Xface; ..valid
* char ch = 0777; ..valid
* char ch = 65535; ..valid
* char ch = 65536; ..invalid

We can represent char literal in Unicode representation which nothing but ‘\uXXXX’ (XXXX = 4 digit hexadecimal number)

* char ch = ‘\u0061’

o/p = a

Every escape character is a valid char literal

* char ch = ‘\n’; ..valid
* char ch = ‘\t’; ..valid
* char ch =’\m’; ..invalid
* Unresolved compilation problem:

Invalid escape sequence (valid ones are \b \t \n \f \r \" \' \\)

|  |  |
| --- | --- |
| Escape Character | Description |
| \n | New line |
| \t | Horizontal tab |
| \r | Carriage return |
| \b | Back space |
| \f | Form feed |
| \’ | Single quote |
| \” | Double quote |
| \\ | Back slash |

***String Literals:***

Any sequence of characters within double quotes is treated as string literal.

E.g : String s = “Darshana”;

**1.7 version Enhancements w.r.t. literals**:

* ***Binary Literals***

For integral data types until 1.6 version we can specify literal value in the following ways:Decimal, Octal, Hexadecimal.

But from 1.7 version we can also specify in Binary form. Allowed digits are 0 and 1.

Literal values should be prefixed with 0b or 0B.

* ***Usage of underscore symbol in numeric literals***

From 1.7 version we can use ‘\_’ symbol between digits of numeric literals.

E.g : double d = 123\_4\_56.7\_8\_9;

The main advantage of this approach is readability of the code will be improved.

At the time of compilation these underscore symbols will be removed automatically. Hence after compilation the above line will become double d = 123456.789;

We can even use more than one underscore symbol between the digits.

E.g : double d = 1\_\_\_2\_3\_4\_\_\_5\_\_6.7\_\_\_\_8\_9;

We can use underscore symbol only between the digits. If we are using it anywhere else we will get compile time error

* double d = \_1\_23\_4\_5.78\_9; ..invalid
* double d = 1\_23\_4\_.7\_89; ..invalid
* double d = = 1\_23\_4.7\_89\_; ..invalid

1b 2b

BYTE SHORT

4b 8b 4b 8b

INT LONG FLOAT DOUBLE

2b

CHAR

8 Byte long value can be assigned to 4 byte float variable because both are following different memory representations internally

E.g : float f = 10L;

Output: 10.0

Both char and short data types are of 2 bytes but short value cannot be stored in char variable and vice versa. This is because short is signed data type and negative values cannot be stored in char variable . And values greater than 32767 cannot be stored in short variable which can be stored in char variable.