Tutorial 1 : Language Fundamentals

* Identifiers
* Reserve Words
* Data Types
* Literals
* Arrays
* Types of Variables
* Var-args Methods
* Main Methods
* Command Line Arguments
* Java Coding Standards

**IDENTIFIERS**

A name in java programming is called Identifier, which can be used for identification purposes. It can be method name or variable name, class name or label name

class Test

{

public static void main (String [] args)

{ int x;

// System.out.print( “ Hello ”);

}

}

Following are the identifiers in above program

* Test – class name
* Main - method name
* String - class name
* args – variable name (name of array)
* X - variable name

Rules for defining java identifiers:

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

Only allowed characters in java identifiers are above

**Rules**:- for identifiers

* If we are using any other character , we will get compile time error

Total\_number – allowed

Total# - not allowed

* Identifiers can’t starts with numerical values

Total123 – valid

123Total – invalid

* Java identifiers are **case sensitive** of course java languages itself is treated as case sensitive

Programming language

class Test

{

int number = 10;

int Number = 20;

int NUMBER = 30;

}

In this case we can differentiate with respect to case.

* In Java , there is no length limit for java identifiers, but it is not recommended to take too lengthy identifiers

Class Test

{

public static void main (String [] args)

{ int xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx = 10;

// System.out.print( “ Hello ”);

System.out.print(xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx);

}

}

This code is a valid code as java doesn’t have any limit on the length of identifiers,

But it is not a good practice to use a lengthy identifiers.

* We can’t use reserved words as identifiers

Int x = 10 - valid;

Int if = 10 – invalid; (because if is reserved words)

We can’t use reserve words as identifiers

* We can use already defined java class name or interface name as identifiers.

class Test{

public static void main (String [] args)

{

int String = 888;

int Runnable = 999;

System.out.println(String);

System.out.println(Runnable);

}

}

* All predefined class name and interface name we can use as a identifiers, but it is not a good practice to use it.
* Eventhough it is valid but it is not a good programming practice , because it reduces readability and creates confusion

Which of the following are valid java identifiers

* Total\_number - valid
* Total# - not valid
* 123total - not valid
* Total123 - valid
* ca$h - valid
* \_$\_$\_$\_$\_ - valid
* all@hands -not valid
* java2share -valid
* Integer - valid – class name
* Int - valid – not a reserve words
* Int - not valid - reserve words

In java some words are reserved words to represent some meaning or functionality such types of words are called reserved words (53) keywords are in java

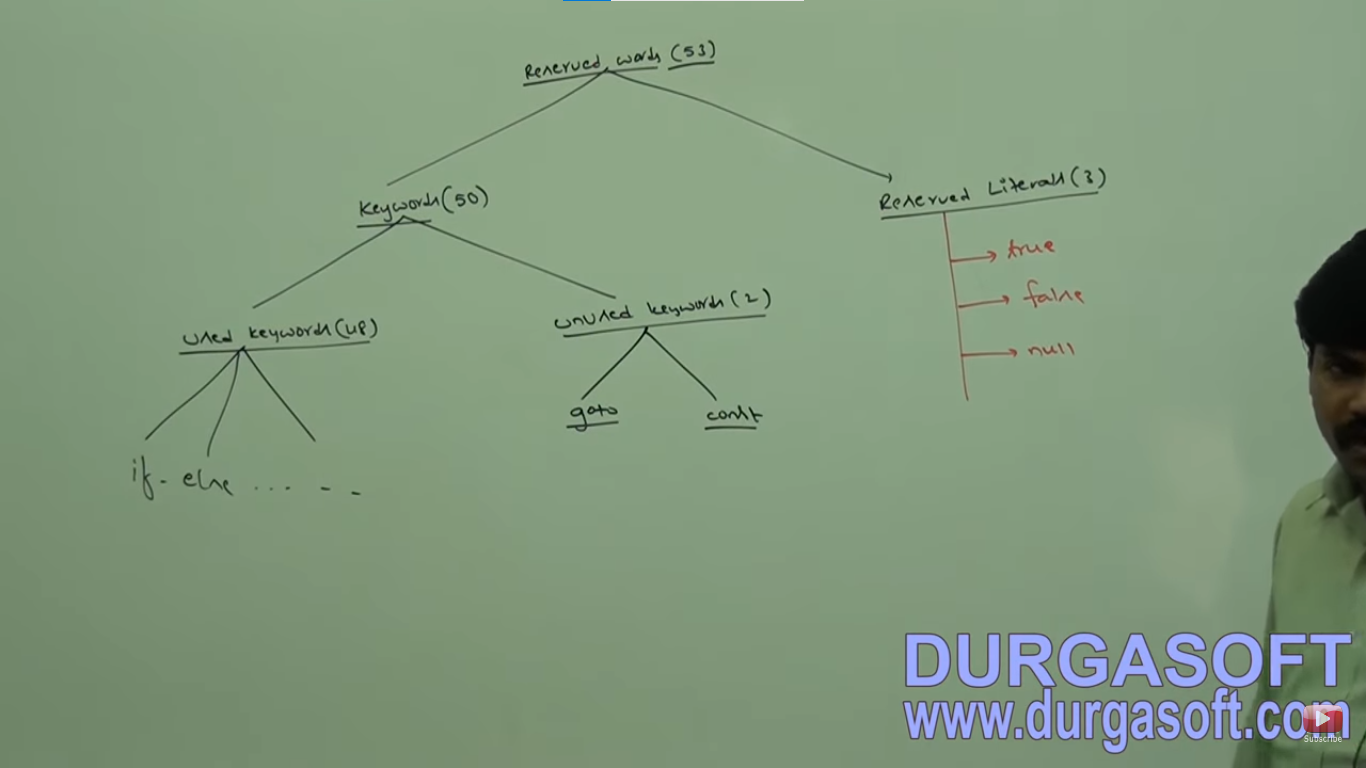
<Goto - 35.12 >

Reserved Words (53) = Keywords (50) + Reserved Literals (3)

Keywords (50) – if the reserved word is assoicated with functionality

Reserved Literals (3) - [true, false, null]

Keyword(50) = used keywords(48) + nounsed keywords(2)



Keywords for dataTypes (8) - bit, short, int, long, float, double, boolean, char

Keywords for flow control (11)- if-else. switch (case, default ), while , do-while, 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

Keyword related to class (6)- class, interface, extends, implements, package, imports

Object related keywords - new , instanceof, super, this,

in Java return type is mandatory, if method don't written anything we have to return this method with void return type

**Unused keywords:** Goto and const - usage of these reserved words give you compile time error

**Reserved Literals:**  true, false, null

true, false - value for default boolean data types

\*\* null - default value for object reference.

Enum keyword : we can use it to defined a group of named constant

**Common conclusion for Keywords:**

1. all 53 keywords contain lower case alphabet and no numerical

2. in java we have only "new" keyword and there is no delete keyword because destruction of useless object is the responsbility of Garbage Collector

3.The following are new keyword in java

strictfp, enum, assert

4. strictfp is not strictFp ,

instanceof but not instanceOf,

synchronized but not synchronize

extends but not extend,

implements but not implements,

import but not imports,

const but not constant

1. Which of the following list contains only java reserved words.

1. new , delete

2. goto, constant

3. break, continue, return, exit

4. final, finally, finalize

5. throw, thorws, thrown

6. notify, notifyAll

7. implements, extends, imports

8. sizeof, instanceof

9. instanceof, strictFp

10. byte, short, Int

11. None of the above - Ans

2. Which of the following is the reserved words

public

static

void

main - name of method

String - name of class

argse - name of variable

**Tutorial - 2**

In java every variable and every expression has some type

Each and every datatype is clearly defined

Every assignement should be checked by compiler for type compatability

Because of above reasons we can conclude , java language strongly typed programming language

Java is pure object oriented language or not ?

1. No, becuase several OOPs feautre is not suported by Java (Operator Overloading , multiple inheritance)

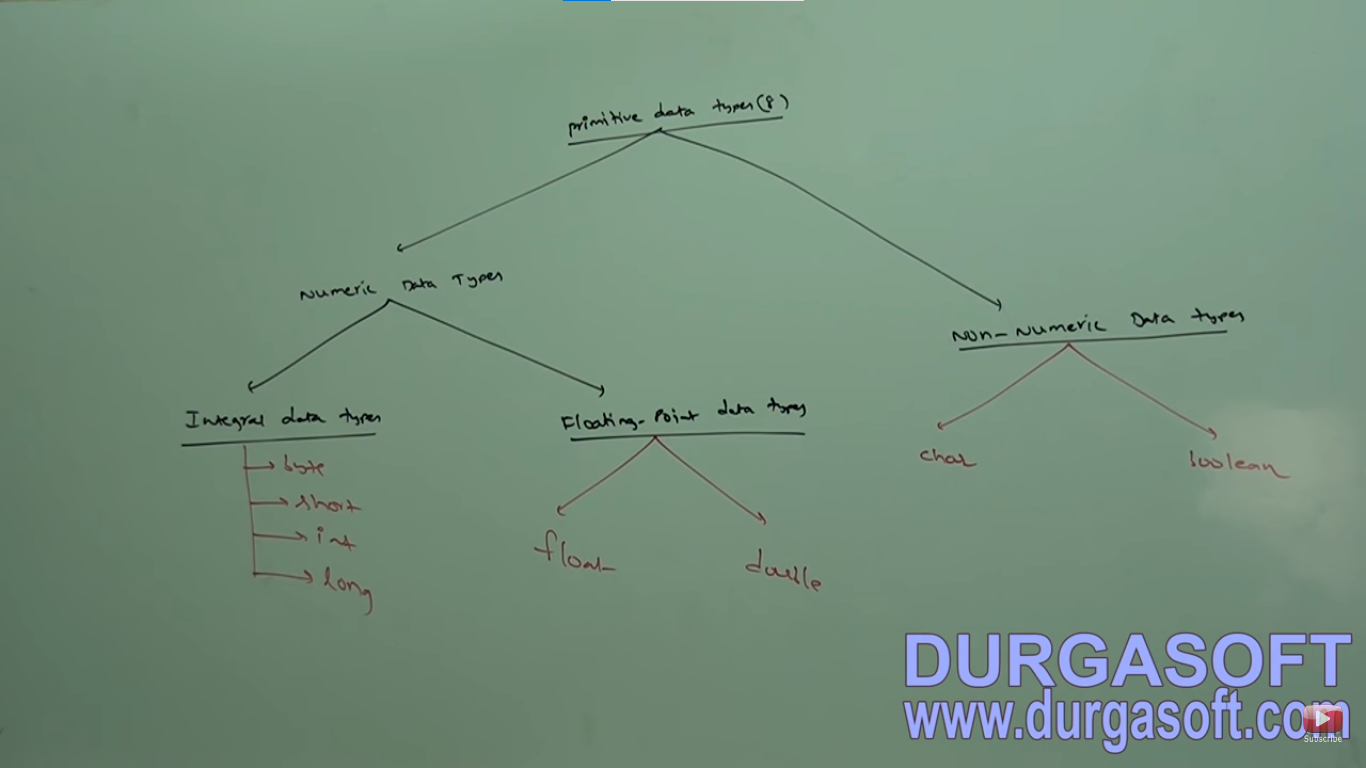
2. Moreover, we are depending on primitive data types, which are non objects

it support primitive data types.

Primitive DataTypes in Java.

Numberical Data Types: - byte, int, short, long, float, double

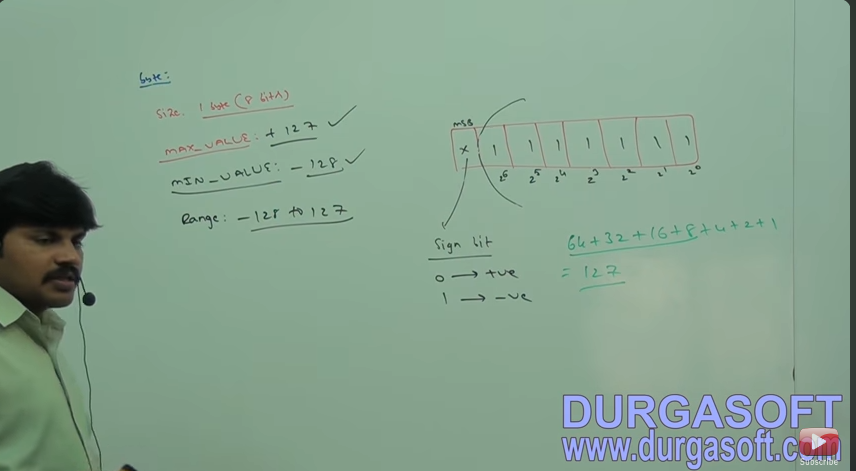
Non Numberical Data Types: boolean



\*\*

Except **boolean and char** remaining data types are considered as - Signed Data Types

int x = -10 or int x = 10; float b = -10.23 or float b = 10.23



the most significate bit is Sign bit .

0 = positive number

1= Negative Number

positive number will be represented directly in memory, negative will be represented in 2' complemented form.

**Tutorial 3:**

[(185) Core Java with OCJP/SCJP: Language Fundamentals Part- 3 || Data Types part-2 - YouTube](https://www.youtube.com/watch?v=HYAosZptldA)

|  |  |
| --- | --- |
| **Float** | **double** |
| Want 4 to 5 decimal of accuracy | Want 14 to 15 decimal of accuracy |
| Single precision | Double precision |
| Size 4 bytes | Size 8 bytes |
|  |  |

Asfj

**\*\*Imp:**

Boolean: Default values are true and false both in **lower case**

**boolean b = true – correct**

**boolean b = 0 – error: incompatible type required Boolean found int**

**boolean b = “True” – error: incompatible type required Boolean found String**

**boolean b = True – cannot find symbol , Symbol: Variable True, location: class Test**

**Ex: int x = 0;**

**If(x)**

**{**

**System.out.println(“Hello”); Compile Error: incompatible type**

**} found int**

**Required boolean**

**Else**

**{**

**System.out.println(“Hello”);**

**}**

Same with While (1)

{

System.out.println(“Hello”);

}

**Char DataTypes:**

Old languages C or C++ ,

The number of allowed ASCII code characters are <=256

To represent this 256 characters 8 bit are enough hence, the size of char in old languages is 1 byte

But java is Unicode base under the number of different Unicode characters are >256 under <65536

To represent this many characters 8 bit may not enough compulsory we should go for 16 bits

Hence the size of char is 2 bytes, better to take 2 bytes a range 0 - 65535

**Summary: Java Primitive Data Types:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Type | Size | Range | Wrapper | Default Value |
| Byte | 1 Byte |  | Byte | 0 |
| Short | 2 Bytes |  | Short | 0 |
| Int | 4 Bytes |  | Integer | 0 |
| Long | 8 Bytes |  | Long | 0 |
| Float | 4 Bytes |  | Float | 0.00 |
| Double | 8 Bytes |  | Double | 0.00 |
| Boolean | NA |  | Boolean | False |
| Char | 2 Byte |  | Character | 0 (represent space character) |
| String or Object |  |  |  | Null |

\*\* - Null is not applicable for primitive

Tutorial 4:

[(186) Core Java with OCJP/SCJP: Language Fundamentals Part-4 || Literals Part-1 - YouTube](https://www.youtube.com/watch?v=Eng8Oi-p2r4)

**Literals:**

Asdf

Int x = 10

Int - datatype/ keyword

X – name of variable/ identifier

10 – constant value / literal

Integral literals

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

decimal literals (base 10 allowed digits are 0-9) ex. Int x = 10

Octal form (base 8 allowed digits are 0-7, literal value should be prefixed with “0”) ex. Int x = 010

HexaDecimal form (base 16 ) allowed digits are 0-9 , a to f ) for extra digits a to f , java is case sensitive language , for extra digit we can use any case for writing

this is one of few areas where java is not case sensitive.

The literal value should be prefixed with 0x or 0X ex. Int x = 0X10;

These are only possible way to specify literal value for integral data types.

Possible Question from Literals

Which of the following declaration are valid.

Int x = 10; - correct

Int x = 0786; - wrong one CE: integer number too large

Int x = 0777; perfectly valid – correct one

Int x = 0XFace; - valid

Int x = 0xBeef – valid

Int x = 0xBeer – not valid – r is out of range of hexadecimal

\*\* imp:

Programmer has choice to specify the value in any format , but the compiler will provide the output only in decimal format

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

Int x = 10; - correct

Long x = 10 L

Int x = 10 L;

There is no direct way to specify byte and short literals explicitly

But indirectly we can specify whenever we are assigning integral literal to the byte variable , and the value within the range , then compiler treat its automatically as byte literal and similar for short literals

Byte b = 10;

Byte b = 127 – allowed

Byte b = 128 – possible loss of precision , found int required byte

Short x = 32767 – allowed

Short x = 32768 - possible loss of precision , found int required short

Floating point literals:

Float f = 123.456 f;

By default every floating point literal is of double type and hence, we can’t assign directly to the float variable

But, we can specify floating point literal as float type, by suffixed with small f or F.

double d = 123.456 D; - correct

we can specify explicity floating by lieteral

double d = 123.456d; possible loss of precision found double required float

double d = 123.456 – decimal

double d = 0123.456 – octal – allowed but representation is in decimal form

double d = 0x123.456 - hexadecimal – not allowed

CE: malformed floating point literals

\*\*we can specify floating point literals only in decimal form and we can’t specify in ocatal or hexadecimal form

Double d = 0786;

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

Double d = 0786 – not allowed

Double d = 0xface - allowed

Double d = 0786.0 –allowed

Double d = 0xface.0 - not allowed

**Tutorial 5:**

As

We can specify char literal as single character within single quotes

Char ch= ‘a’;

Char ch = a; here compiler will treat ‘a’ as a variable

Char ch = “a” – CE: incompatible type found String required char.

Char ch = ‘ab’ - CE: unclosed character literal

Char ch = 97;

Sop(ch); //a

The possible range allowed is the 0 to 65536

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

Char ch = 0xface - allowed

Char ch = 0777; - allowed

Char ch = 655535 – allowed

Char ch = 655536; CE:// possible loss of precision found int required character

We can represent char literal in Unicode representation which is nothing but ‘\uxxx’

Char ch = ‘\u0061’;

Sop(ch);

Ch = a;

Escape character

‘/n’ = new line

‘/t’ = tab

‘/m’ = CE: illegal escape character

Every escape character is a valid char literal

Escape character | description

\n | new line

\t | new tab

` \r carriage return

\b Back space

\f from feed

\’ single quote

\” double quote

\\ Back slash

Which of the following are valid

Char ch = 65536; - invalid – max range is 65535

Char ch = 0xBeer; - invalid - r is out of range

Char ch = \uFace; - invalid single quote missing

Char ch = ‘\ubeef’; - valid

Char ch = ‘\m’; - not valid

Char ch = ‘\iface’; - not valid

**String literal:**

Char ch = ‘\iface’; - not valid

Any sequence of character within double quotes is treated as String literal

String s =

1.7 version enhancement with respect to literals

1. Binary literals – int x = 0B 1111,

2.

For integral datatypes unit 1.6 version we can specify literal value in the following ways

Decimal form , octal form, hexadecimal form ,

From the 1.7 version onwards we can specify literal value even in binary form also.

Allowed digits are 0 and 1;

Literal value should be prefixed with 0b or 0B

Double d = 123456.789

From 1.7 version onward we can use \_ symbols between digits of numeric literal

Double d = 1\_23\_456.7\_8\_9;

as well we can write like

Doubled d = 123\_456.7\_8\_9

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

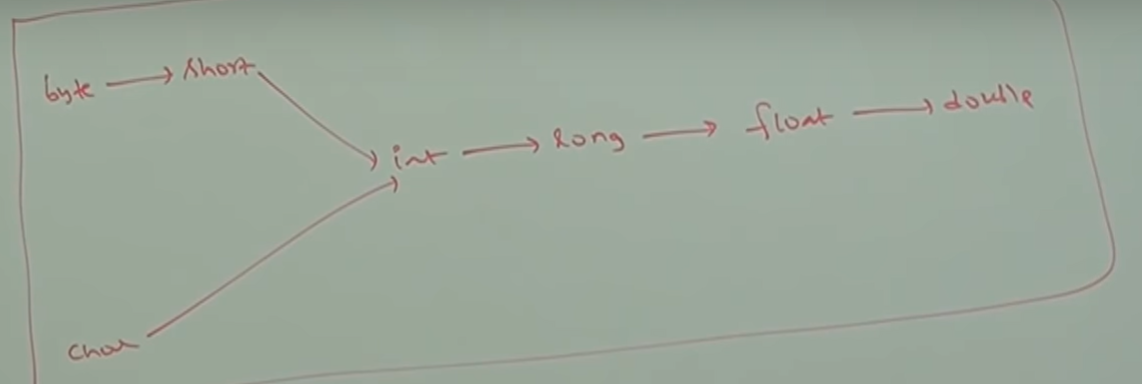
We can use more than one “ \_” symbol also between the digits

Double d = \_1\_23\_456.789; - invalid

Double d = \_1\_ \_23\_ \_ \_456.789\_; - invalid

We can use \_ only between the digits if we are using anywhere else we wil get compile time error

Summary Point:



This type of assignment of variable is always acceptable

8 byte long value we can assign to 4 byte float variables because both are following different memory representation internally

Float f = 10l;

Sop(f); // 10.0