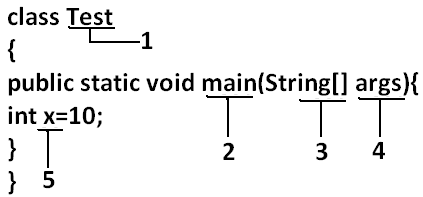
**Identifier:**A name in java program is called identifier. It may be class name, method name, variable name and label name.

**Example:**



**Rules to define java identifiers:**

**Rule1:** The only allowed characters in java identifiers are:

1. a to z
2. A to Z
3. 0 to 9
4. \_
5. $

**Rule 2:**If we are using any other character we will get compile time error.

**Example:**

1. total\_number-------valid
2. Total#------------------invalid

**Rule 3:**identifiers are not allowed to starts with digit.

**Example:**

1. ABC123---------valid
2. 123ABC---------invalid

**Rule 4:**java identifiers are case sensitive up course java language itself treated as case sensitive language.

**Example:**

class Test{

int number=10;

int Number=20;

int NUMBER=20; we can differentiate with case.

int NuMbEr=30;

}

**Rule 5:**There is no length limit for java identifiers but it is not recommended to take more than 15 lengths.

**Rule 6:** We can’t use reserved words as identifiers.

**Example:** int if=10; --------------invalid

**Rule 7:** All predefined java class names and interface names we use as identifiers.

**Example 1:**

class Test

{

public static void main(String[] args){

int String=10;

System.out.println(String);

}}

**Output:**

10

**Example 2:**

class Test

{

public static void main(String[] args){

int Runnable=10;

System.out.println(Runnable);

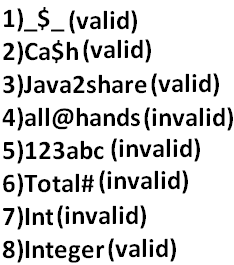
}}

**Output:**

10

Even though it is legal to use class names and interface names as identifiers but it is not a good programming practice.

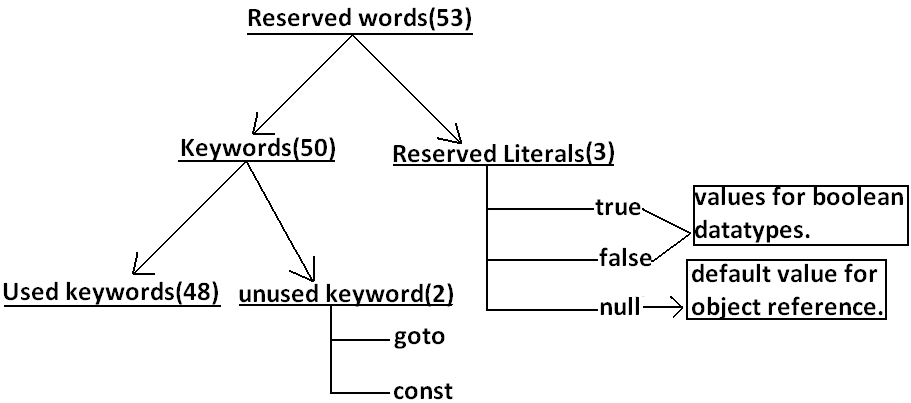
**Which of the following are valid java identifiers?**



**Reserved words:**

* In java some identifiers are reserved to associate some functionality or meaning such type of reserved identifiers are called reserved words.

**Diagram:**

****

**Reserved words for data types:**

1. byte
2. short
3. int
4. long
5. float
6. double
7. char
8. boolean

**Reserved words for flow control:**

1. if
2. else
3. switch
4. case
5. default
6. for
7. do
8. while
9. break
10. continue
11. return

**Keywords for modifiers:**

1. public
2. private
3. protected
4. static
5. final
6. abstract
7. synchronized
8. native
9. strictfp(1.2 version)
10. transient
11. volatile

**Keywords for exception handling:**

1. try
2. catch
3. finally
4. throw
5. throws
6. assert(1.4 version)

**Class related keywords:**

1. class
2. package
3. import
4. extends
5. implements
6. interface

**Object related keywords:**

1. new
2. instanceof
3. super
4. this

**Void return type keyword:**

* If a method won’t return anything compulsory that method should be declared with the void return type in java but it is optional in C++.

1. void

**Unused keywords:**

**goto**:Create several problems in old languages and hence it is banned in java.

**Const:**Use final instead of this.

* By mistake if we are using these keywords in our program we will get compile time error.

**Reserved literals:**

1. true values for boolean data type.
2. false
3. null----------------- default value for object reference.

**Enum:**

* This keyword introduced in 1.5v to define a group of named constants

**Example:**

enum Beer

{

KF, RC, KO, FO;

}

**Note:**All reserved words in java contain only lowercase alphabet symbols.

New keywords are:

Strictfp-----------1.2

Assert-------------1.4

Enum--------------1.5

**Which of the following list contains only java reserved words?**

1. final, finally,finalize (invalid)//here finalize is a method in Object class.
2. throw, throws, thrown(invalid)//thrown is not available in java
3. break, continue, return, exit(invalid)//exit is not reserved keyword
4. goto, constant(invalid)//here constant is not reserved keyword
5. byte, short, Integer, long(invalid)//here Integer is a wrapper class
6. extends, implements, imports(invalid)//imports keyword is not available in java
7. finalize, synchronized(invalid)//finalize is a method in Object class
8. instanceof, sizeOf(invalid)//sizeOf is not reserved keyword
9. new, delete(invalid)//delete is not a keyword
10. None of the above(valid)

**Which of the following are valid java keywords?**

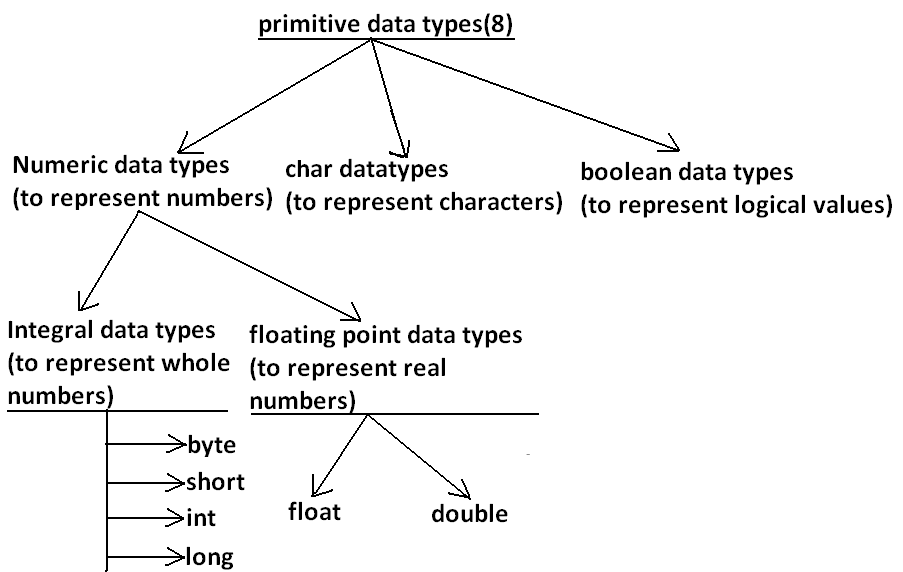
1. public(valid)
2. static(valid)
3. void(valid)
4. main(invalid)
5. String(invalid)
6. args(invalid)

**Data types:**Every variable has a type, every expression has a type and all types are strictly define more over every assignment should be checked by the compiler by the type compatibility hence java language is considered as strongly typed language.

**Java is pure object oriented programming or not?**

* Java is not considered as pure object oriented programming language because several oops features (like multiple inheritance, operator overloading) are not supported by java moreover we are depending on primitive data types which are non objects.

**Diagram:**

****

* Except Boolean and char all remaining data types are considered as signed data types because we can represent both “+ve” and”-ve” numbers.

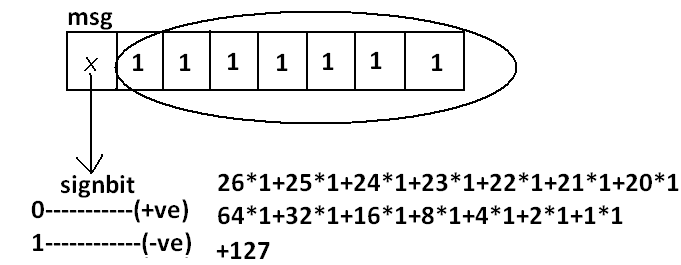
**Byte:**

Size: 1byte (8bits)

Maxvalue: +127

Minvalue:-128

Range:-128to 127[-27 to 27-1]



* The most significant bit acts as sign bit. “0”means “+ve” number and “1” means “–ve” number.
* “+ve” numbers will be represented directly in the memory whereas “–ve” numbers will be represented in 2’s complement form.

**Example:**

byte b=10;

byte b2=130;//C.E:possible loss of precision

byte b=10.5;//C.E:possible loss of precision

byte b=true;//C.E:incompatible types

byte b="ksrao";//C.E:incompatible types

* byte data type is best suitable if we are handling data in terms of streams either from the file or from the network.

**Short:**

* The most rarely used data type in java is short.

Size: 2 bytes

Range: -32768 to 32767(-215 to 215-1)

**Example:**

short s=130;

short s=32768;//C.E:possible loss of precision

short s=true;//C.E:incompatible types

* Short data type is best suitable for 16 bit processors like 8086 but these processors are completely outdated and hence the corresponding short data type is also out data type.

**Int:**

* This is most commonly used data type in java.

Size: 4 bytes

Range:-2147483648 to 2147483647 (-231 to 231-1)

**Example:**

int i=130;

int i=10.5;//C.E:possible loss of precision

int i=true;//C.E:incompatible types

**long:**

* Whenever int is not enough to hold big values then we should go for long data type.

**Example:**

* To hold the no. Of characters present in a big file int may not enough hence the return type of length() method is long.

long l=f.length();//f is a file

Size: 8bytes

Range:-263to 263-1

**Note:**All the above data types (byte, short, int and long) can be used to represent whole numbers. If we want to represent real numbers then we should go for floating point data types.

**Floating Point Data types:**

|  |  |
| --- | --- |
| **Float** | **double** |
| 1. If we want to 5to 6 decimal places of accuracy then we should go for float. | 1. If we want to 14 to 15 decimal places of accuracy then we should go for double. |
| 1. Size:4 bytes. | 1. Size:8 bytes. |
| 1. Range:-3.4e38 to3.4e38. | 1. -1.7e308 to1.7e308. |
| 1. float follows single precision. | 1. double follows double precision. |

**boolean data type:**

Size: Notapplicable (virtual machine dependent)

Range: Not applicable but allowed values are true or false.

**Which of the following boolean declarations are valid?**

**Example 1:**

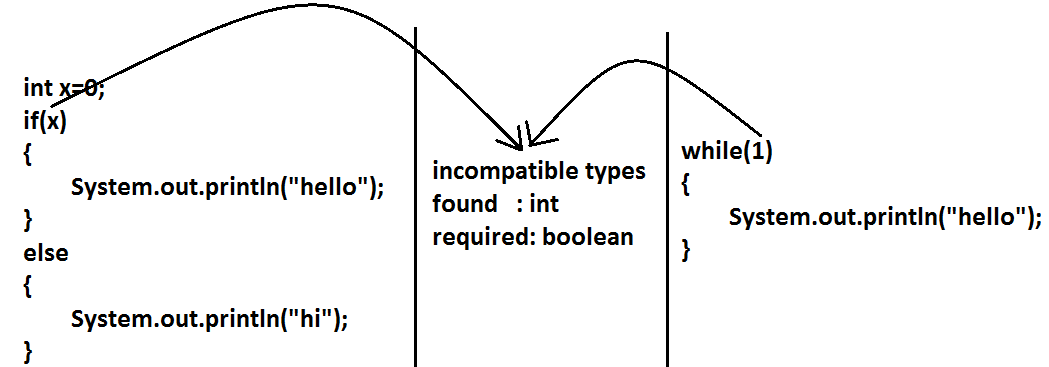
boolean b=true;

boolean b=True;//C.E:cannot find symbol

boolean b="True";//C.E:incompatible types

boolean b=0;//C.E:incompatible types

**Example 2:**



**Char data type:**

* In java we are allowed to use any worldwide alphabets character and java is Unicode based to represent all these characters one byte is not enough compulsoryweshould go for 2 bytes.

**Size: 2 bytes**

**Range: 0 to 65535**

**Example:**

char ch1=97;

char ch2=65536;//C.E:possible loss of precision

**Summary of java primitive data type:**

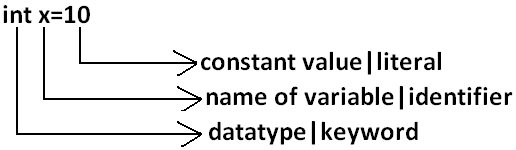
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **data type** | **Size** | **Range** | **Corresponding**  **Wrapper class** | **Default value** |
| Byte | 1 byte | -27 to 27-1  (-128 to 127) | Byte | 0 |
| Short | 2bytes | -215 to 215-1  (-32768 to 32767) | Short | 0 |
| Int | 4 bytes | -231 to 231-1  (-2147483648 to 2147483647) | Integer | 0 |
| Long | 8 bytes | -263 to 263-1 | Long | 0 |
| Float | 4 bytes | -3.4e38 to 3.4e38 | Float | 0.0 |
| Double | 8 bytes | -1.7e308 to 1.7e308 | Double | 0.0 |
| Boolean | Not applicable | Not applicable(but allowed values true|false) | Boolean | false |
| Char | 2 bytes | 0 to 65535 | Character | 0(represents blank space) |

* The default value for the object references is “null”.

**Literals:**

* Any constant value which can be assigned to the variable is called literal.

**Example:**



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

1. **Decimal literals:**Allowed digits are 0 to 9.

**Example:** int x=10;

1. **Octal literals:**Allowed digits are 0 to 7. Literal value should be prefixed with zero.

**Example:**int x=010;

1. **Hexa Decimal literals:**The allowed digits are 0 to 9, A to Z.For the extra digits we can use both upper case and lower case characters. This is one of very few areas where java is not case sensitive. Literal valueshouldbe prefixed with ox(or)oX.

**Example:** int x=0x10;

* These are the only possible ways to specify integral literal.

**Which of the following are valid declarations?**

1. int x=0786;//C.E:integer number too large: 0786(invalid)
2. int x=0xFACE;(valid)
3. int x=0xbeef;(valid)
4. int x=0xBeer;//C.E:';' expected(invalid)

//:int x=0xBeer;

^ // ^

1. int x=0xabb2cd;(valid)

**Example:**

int x=10;

int y=010;

int z=0x10;

System.out.println(x+"----"+y+"----"+z); //10----8----16

* By default every integral literal is int type but we can specify explicitly as long type by suffixing with small “**l**” (or) capital“**L**”.

**Example:**

int x=10;(valid)

long l=10L;(valid)

long l=10;(valid)

int x=10l;//C.E:possible loss of precision(invalid)

* There is no direct way to specify byte and short literals explicitly. But whenever we are assigning integral literal to the byte variables and its value within the range of byte compiler automatically treats as byte literal. Similarly short literal also.

**Example:**

byte b=10;(valid)

byte b=130;//C.E:possible loss of precision(invalid)

short s=32767;(valid)

short s=32768;//C.E:possible loss of precision(invalid)

**Floating Point Literals:** Floating point literal is by default double type but we can specify explicitly as float type by suffixing with f or F.

**Example:**

float f=123.456;//C.E:possible loss of precision(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.

**Example:**

double d=123.456D;

* We can specify floating point literal only in decimal form and we can’t specify in octal and hexadecimal forms.

**Example:**

double d=123.456;(valid)

double d=0123.456;(valid)

double d=0x123.456;//C.E:malformed floating point literal(invalid)

**Which of the following floating point declarations are valid?**

1. float f=123.456;//C.E:possible loss of precision(invalid)
2. float f=123.456D;//C.E:possible loss of precision(invalid)
3. double d=0x123.456;//C.E:malformed floating point literal(invalid)
4. double d=0xFace;(valid)
5. double d=0xBeef;(valid)

* We can assign integral literal directly to the floating point data types and that integral literal can be specified in octal and Hexa decimal form also.

**Example:**

double d=0xBeef;

System.out.println(d);//48879.0

* But we can’t assign floating point literal directly to the integral types.

**Example:**

int x=10.0;//C.E:possible loss of precision

* We can specify floating point literal even in exponential form also(significant notation).

**Example:**

double d=10e2;//==>10\*102(valid)

System.out.println(d);//1000.0

float f=10e2;//C.E:possible loss of precision(invalid)

float f=10e2F;(valid)

**Boolean literals:** The only allowed values for the boolean type are true (or) false where case is important.

**Example:**

1. boolean b=true;(valid)
2. boolean b=0;//C.E:incompatible types(invalid)
3. boolean b=True;//C.E:cannot find symbol(invalid)
4. boolean b="true";//C.E:incompatible types(invalid)

**Char literals:**

1. A char literal can be represented as single character within single quotes.

**Example:**

1. char ch='a';(valid)
2. char ch=a;//C.E:cannot find symbol(invalid)
3. char ch="a";//C.E:incompatible types(invalid)
4. char ch='ab';//C.E:unclosed character literal(invalid)
5. We can specify a char literal as integral literal which represents Unicode of that character. We can specify that integral literal either in decimal or octal or hexadecimal form but allowed values range is 0 to 65535.

**Example:**

1. char ch=97;(valid)
2. char ch=0xFace; (valid)

System.out.println(ch);//?

1. char ch=65536;//C.E: possible loss of precision(invalid)
2. We can represent a char literal by Unicode representation which is nothing but ‘\uxxxx’.

**Example:**

1. char ch1='\u0061';

System.out.println(ch1);//a

1. char ch2=\u0062;//C.E:cannot find symbol
2. char ch3='\iface';//C.E:illegal escape character
3. Every escape character in java acts as a char literal.

**Example:**

1. char ch='\n';//(valid)
2. char ch='\l';//C.E:illegal escape character(invalid)

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

**Which of the following char declarations are valid?**

1. char ch=a;//C.E:cannot find symbol(invalid)
2. char ch='ab';//C.E:unclosed character literal(invalid)
3. char ch=65536;//C.E:possible loss of precision(invalid)
4. char ch=\uface;//C.E:illegal character: \64206(invalid)
5. char ch='/n';//C.E:unclosed character literal(invalid)
6. none of the above.(valid)

**String literals:**

* Any sequence of characters with in double quotes is treated as String literal.

**Example:**

String s="bhaskar";(valid)

**Diagram:**

