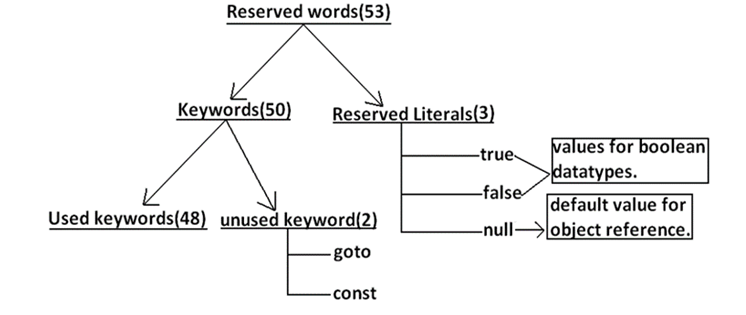
**Language Fundamentals**

**Identifiers:**

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

**Reserved Words:**

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



**Reserved words for data types: (8)**

byte, short, int, long, float, double, char, boolean

**Reserved words for flow control:(11)**

if, else, switch, case, default, for, do, while, break, continue, return

**Keywords for modifiers:(11)**

public,private,protected,static,final,abstract,synchronized,native,strictfp(1.2 version), transient, volatile

**Keywords for exception handling:(6)**

try, catch, finally, throw, throws, assert(1.4 version)

**Class related keywords:(6)**

class, package, import, extends, implements, interface

**Object related keywords:(4)**

new, instanceof, super, this, void , enum

**Note:**

1)All reserved words in java contain only lowercase alphabet symbols

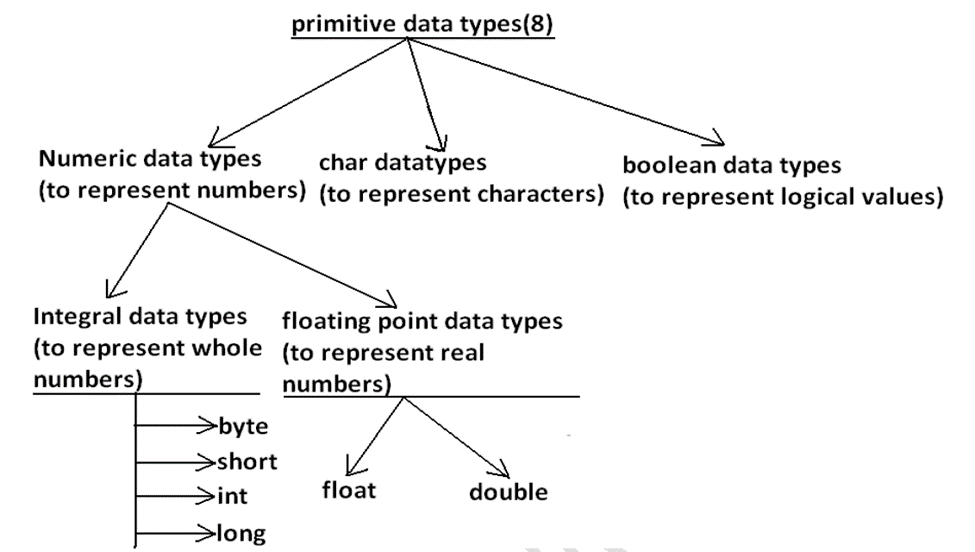
2)In java we have only new keyword but not delete because destruction of useless

objects is the responsibility of Garbage Collection

**Datatypes:**

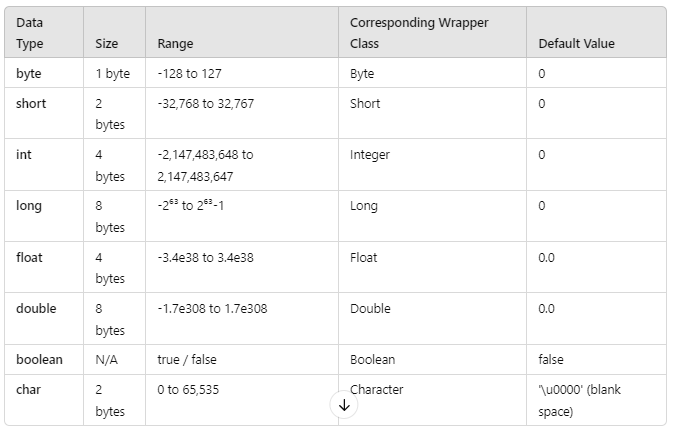
Every variable has a type, every expression has a type and all types are strictly define more over every assignment should be checked by the compiler by the type compatibility hence java language is considered as strongly typed programming language.

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.



**Note :** Except Boolean and char all remaining data types are considered as signed data types

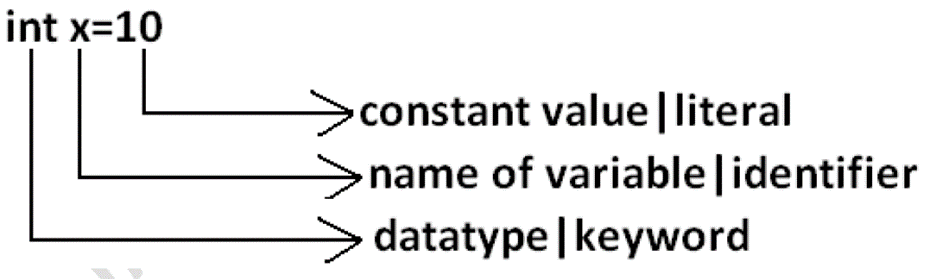
because we can represent both "+ve" and"-ve" numbers.

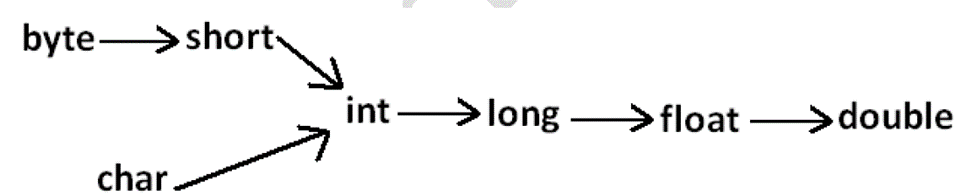


**Literals:**

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

Example:

****

****

**Arrays:**

An array is an indexed collection of fixed number of homogeneous data elements. The main advantage of arrays is we can represent multiple values with the same name so that readability of the code will be improved.

**Main disadvantage of arrays :**

Fixed in size that is once we created an array there is no chance of increasing or decreasing the size based on our requirement that is to use arrays concept compulsory we should know the size in advance which may not possible always. We can resolve this problem by using collections.

**Array Declaration:**

int[] a;

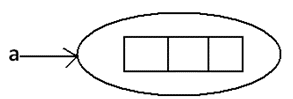
**Note :** At the time of declaration we can't specify the size otherwise we will get compile time error.

**Array construction:**

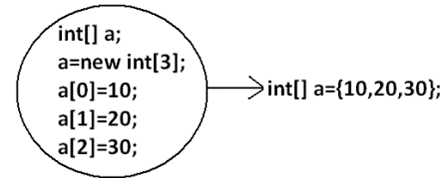
Every array in java is an object hence we can create by using new operator.

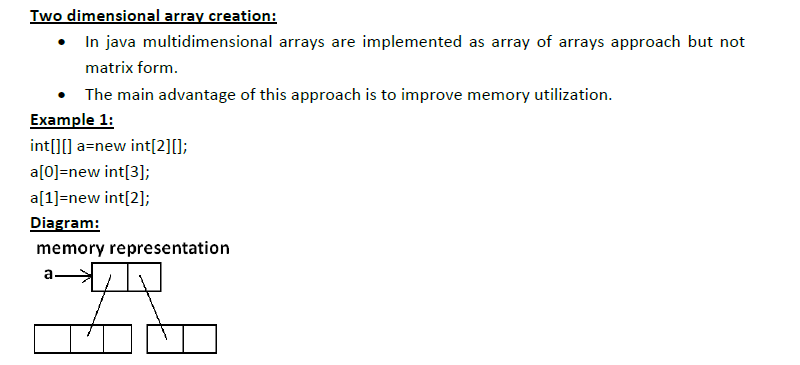
**Example:**

int[] a=new int[3];



**Declaration, construction and initialization of an array in a single line:**

****

****

**length vs length() method:**

length:

1. It is the final variable applicable only for arrays.

2. It represents the size of the array.

**Example:**

int[] x=new int[3];

System.out.println(x.length());//C.E: cannot find symbol

System.out.println(x.length);//3

length() method:

1. It is a final method applicable for String objects.

2. It returns the no of characters present in the String.

**Example:**

String s="bhaskar";

System.out.println(s.length);//C.E:cannot find symbol

System.out.println(s.length());//7

**Types of Variables:**

**Division 1:** Based on the type of value represented by a variable all variables are divided into 2 types. They are:

1. Primitive variables

2. Reference variables

**Primitive variables:**

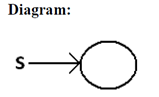
Primitive variables can be used to represent primitive values.

Example: int x=10;

**Reference variables:**

Reference variables can be used to refer objects.

Example: Student s=new Student();



**Division 2 :** Based on the behavior and position of declaration all variables are divided

into the following 3 types.

1. Instance variables

2. Static variables

3. Local variables

**Instance variables:**

· If the value of a variable is varied from object to object such type of variables are

called instance variables.

· For every object a separate copy of instance variables will be created.

· Instance variables will be created at the time of object creation and destroyed at

the time of object destruction hence the scope of instance variables is exactly

same as scope of objects.

· Instance variables will be stored on the **heap** as the part of object.

· Instance variables should be declared with in the class directly but outside of any

method or block or constructor.

· Instance variables can be accessed directly from Instance area. But cannot be

accessed directly from static area.

· But by using object reference we can access instance variables from static area.

· Instance variables also known as object level variables or attributes.

**Example:**

class Test

{

int i=10;

public static void main(String[] args)

{

//System.out.println(i);

//C.E:non-static variable i cannot be referenced from a static context(invalid)

Test t=new Test();

System.out.println(t.i);//10(valid)

t.methodOne();

}

public void methodOne()

{

System.out.println(i);//10(valid)

}

}

**Static variables:**

· If the value of a variable is not varied from object to object such type of variables is not recommended to declare as instance variables. We have to declare such type of variables at class level by using static modifier.

· In the case of instance variables for every object a separate copy will be created but in the case of static variables for entire class only one copy will be created and shared by every object of that class.

· **Static variables will be created at the time of class loading** and destroyed at the time of class unloading hence the scope of the static variable is exactly same as the scope of the .class file.

· Static variables will be stored in **method area**. Static variables should be declared with in the class directly but outside of any method or block or constructor.

· Static variables can be accessed from both instance and static areas directly.

· We can access static variables either by class name or by object reference but usage of class name is recommended.

· But within the same class it is not required to use class name we can access

directly.

· For the static variables it is not required to perform initialization explicitly, JVM will

always provide default values.

**java TEST:**

1. Start JVM.

2. Create and start Main Thread by JVM.

3. Locate(find) Test.class by main Thread.

4. Load Test.class by main Thread. // static variable creation

5. Execution of main() method.

6. Unload Test.class // static variable destruction

7. Terminate main Thread.

8. Shutdown JVM.

**Example:**

class Test

{

static int i=10;

public static void main(String[] args)

{

Test t=new Test();

System.out.println(t.i);//10

System.out.println(Test.i);//10

System.out.println(i);//10

}

}

**Local variables:**

· Some times to meet temporary requirements of the programmer we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.

· Local variables will be stored inside **stack**.

· The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly same as scope of the block in which we declared.

· The local variables will be stored on the stack.

· For the local variables JVM won't provide any default values compulsory we should perform initialization explicitly before using that variable.

**Example 1:**

class Test

{

public static void main(String[] args)

{

int i=0;

for(int j=0;j<3;j++)

{

i=i+j;

}}}

**Var- arg methods (variable no of argument methods) (1.5) :**

· Until 1.4v we can't declared a method with variable no. Of arguments.

· If there is a change in no of arguments compulsory we have to define a new method.

· This approach increases length of the code and reduces readability.

· But from 1.5 version onwards we can declare a method with variable no. Of arguments such type of methods are called var-arg methods.

**Example:**

class Test

{

public static void methodOne(int... x)

{

System.out.println("var-arg method");

}

public static void main(String[] args)

{

methodOne();

methodOne(10);

methodOne(10,20,30);

}

}

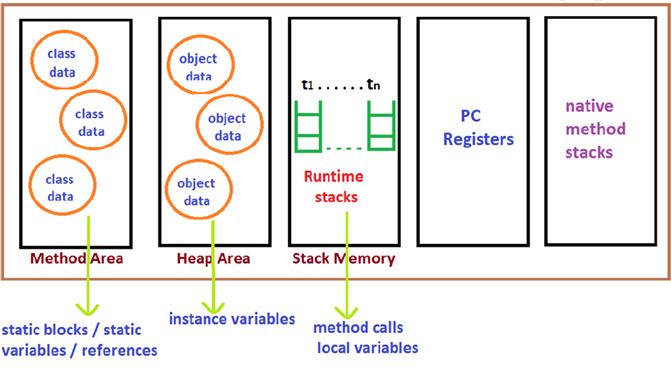
**Output:**

var-arg method

var-arg method

var-arg method

**Various memory areas of JVM:**

****

1. Class level binary data including static variables will be stored in method area.

2. Objects and corresponding instance variables will be stored in Heap area.

3. For every method the JVM will create a Runtime stack all method calls performed by that Thread and corresponding local variables will be stored in that stack. Every entry in stack is called Stack Frame or Action Record.

4. The instruction which has to execute next will be stored in the corresponding PC

Registers.

5.Native method invocations will be stored in native method stacks.

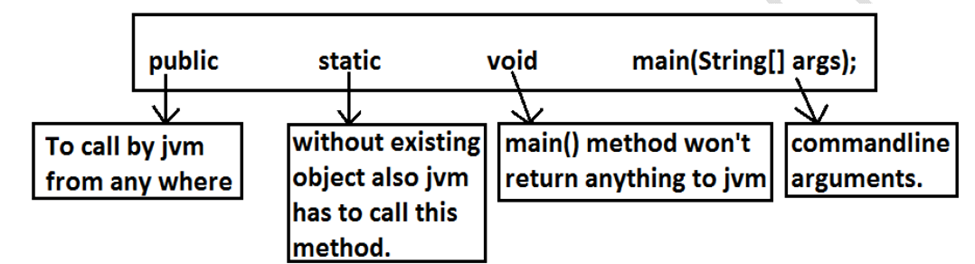
**Main Method:**

Whether the class contains main() method or not, and whether it is properly declared or not, these checking's are not responsibilities of the compiler, at runtime JVM is responsible for this.

If JVM unable to find the required main() method then we will get runtime exception

saying NoSuchMethodError: main.

At runtime JVM always searches for the main() method with the following prototype.



**Note** : The order of modifiers is not important that is instead of public static we can

take static public.

**Java Coding Standards**