Language Fundamentals

Topics covered

1. Identifiers
2. Reserved words
3. Data types
4. Literals
5. Arrays
6. Types of variables
7. var-arg method
8. main () method
9. Command line arguments
10. Java coding standards

1. Identifier

> A name in java program is called Identifier which can be used for identification purpose.

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

For E.g.

```
Class Test {
      public static void main (String[] args) {
            int x = 10;
      }
}
```

Q. How many identifiers are there in above program? -> 5 (Identifiers in bold & italic letter)

> Rules for defining Java Identifiers

a) Allowed characters in java identifiers are a to z, A to Z, 0 to 9, $, _

b) Identifiers can't start with digit.

c) Identifiers are case sensitive. (Java language is itself treated as a case – sensitive programming language.)

d) No length limit for java identifiers but recommended to take simple & understandable name.

e) We can't use reserved words as identifiers.

f) All predefined java class names & interface names can be used as identifiers but not recommended to use that.

2. Reserved words

> In java, some words are reserved to represent special meaning or functionality; such types of words are called **reserved words.**

> Total 53 reserved words in which 50 are keywords & 3 are reserved literals (true, false, null).

> In 50 keywords, 48 are used keywords & 2 are unused (goto, const).

> Use 'final' keyword instead of 'const'.

> In java return type is mandatory. If a method won't return anything then it has to be declared with void return type.

Keywords for Datatype	Keywords for flow control	Keywords for Modifiers
1. byte 2. short 3. int 4. long 5. float 6. double 7. boolean 8. char	1. if 2. else 3. switch 4. case 5. default 6. while 7. do 8. for 9. break 10. continue 11. return	1. public 2. private 3. protected 4. static 5. final 6. abstract 7. synchronized 8. native 9. strictfp 10. transient 11. volatile
Keywords for exception handling 1. try 2. catch 3. finally 4. throws 5. assert 6. throw	Class related keywords 1. class 2. interface 3. extends 4. implements 5. package 6. import Enum keyword (can be used to define a group of named constants) 1. enum	Object related keywords 1. new 2. instanceof 3. super 4. this Returntype keyword 1. void

Fig: Keywords collection used in Java

Note:

> All 53 reserved words in Java contains only lowercase alphabet symbol.

> In Java, we have only new keyword; no delete keyword as destruction of useless object is the responsibility of Garbage collector.

3. Data types

> In java, every variable & every expression has some type.

> Each & every datatype is clearly defined.

> Every assignment should be checked by compiler for type compatibility because of this reason; we can conclude Java language is strongly typed programming language.

> Java is **not considered as pure Object oriented programming** language because several OOP's features are not satisfied by Java (like operator overloading, multiple inheritance etc.) and moreover we're depending on primitive datatypes which are non – objects.

> Except Boolean & char, remaining datatypes are considered as signed datatypes.

Primitive datatype (8)	
Numeric datatype	Non-Numeric datatype
1. byte 2. short 3. int 4. long 5. float 6. double	1. char 2. boolean

> Summary of Java Primitive Data types

No.	Datatype	Size	Range	Wrapper class	Default Values
1.	byte	1 byte	-2^7 to $2^7 - 1$	Byte	0
2.	short	2 bytes	-2^{15} to $2^{15} - 1$	Short	0
3.	int	4 bytes	-2^{31} to $2^{31} - 1$	Integer	0
4.	long	8 bytes	-2^{63} to $2^{63} - 1$	Long	0
5.	float	4 bytes	$-3.4e^{38}$ to $3.4\ e^{38}$	Float	0.0f
6.	double	8 bytes	$-1.7e^{308}$ to $1.7\ e^{308}$	Double	0.0d
7.	boolean	NA	NA	Boolean	false
8.	char	2 bytes	0 to 65535	Character	'\u0000'

Note:

> The most significant bit (MSB) is sign bit. 0 means positive no. & 1 means negative no.

> Positive Numbers will be represented directly in the memory whereas Negative numbers will be represented in 2's Complement form.

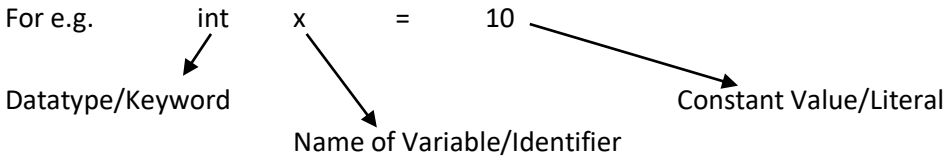
> Byte is the best choice if we want to handle data in terms of Streams either from the file or from the network i.e. File/Network Supported Form is byte.

> Old languages (C, C++) are ASCII code based & no of different allowed ASCII characters are less than 256 that's why the size of char in old languages is 1 byte whereas Java is Unicode based & the no. of different Unicode characters are greater than 256 but less than 65536. Hence the size of char in java is 2 bytes.

> Default value for Object reference is null.

4. Literals

> A constant value which can be assigned to the variable is called as the literal.



a) Integral Literals

> For Integral datatypes (byte, short, int, long) we can specify literal values in the following ways :

1. Decimal literal/form (base 10) – allowed digits are 0 -> 9
2. Octal form (base 8) – allowed digits are 0 -> 7; Also literal values should be prefixed with zero (0)
   E.g. int x = 010;
3. Hexadecimal form (base 16) – allowed digits are 0 -> 9 , a -> f or A -> F; Also literal values should be prefixed with 0x or 0X (zero x).
   E.g. int x = 0X10;
4. Binary form (base 2) – allowed digits are 0 & 1; Also literal values should be prefixed with 0b or 0B (zero b).
   E.g. int x = 0B1111;

> Programmers have a choice to specify the value in any forms but JVM will always provide the value only in decimal form.

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

> There is no direct way to specify byte & short literals explicitly but indirectly we can specify. Whenever we're assigning integral literal to a byte/short variable, if the value is within the range of byte/short, Compiler automatically treats it as byte/short literal.

b) Floating point Literals

> By default, every floating point literal is of double type & hence we can't assign directly to the float variable. But we can specify floating point literal as float type by suffixing the literal with 'f' or 'F'.

E.g. float f = 123.35F; double d = 123.32

> We can specify floating point literals only in decimal form & exponential form. But we can assign integral literal (present in any form) directly to floating point variables.

E.g. double d = 0XFace; double d = 0785.0; double d = 1.2 e^3 -> 1200.0

c) Boolean Literals

> The only allowed values for Boolean datatype are true & false.

d) Char Literals

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

E.g. char ch = 'a';

> We can specify char literal as integral literal (in any form) which represents Unicode of the character but allowed range is 0 -> 65535

E.g. char ch = 0XFace; char ch = 65535;

> We can represent char literal in Unicode representation i.e. '\uxxxx' (4 digit hexadecimal no.)

E.g. char ch = '\u0061'; -> a

> Every escape character is a valid char literal.

E.g. char ch = '\n'; char ch = '\t'; char ch = '\\';

e) String Literal

> Any sequence of characters within double quote ("") is treated as String literal.

E.g. String str = "durga";

Note:

> We can use '_' (underscore) between digits of numeric literals.

E.g. double d = 1_23_456.7_8_9;

> At the time of compilation, these _ symbols will be removed automatically.

> 8 byte long value can be assigned to 4 byte float variable because both are following different memory representation internally. E.g. float f = 10L;

> But 2 byte short can't be assigned to 2 byte char as both are different with respect to sign. In short, only 15 bits represent value, 1 bit is for sign.

5. Arrays

> An array is an indexed collection of fixed no. of homogenous data elements.

> The main advantage of arrays is we can represent huge no. of values by using single variable so that readability of the code will be improved.

> But the main disadvantage of arrays is:

> Fixed in size i.e. once we create an array, there is no chance of increasing or decreasing the size based on our requirement. Hence to use array concept, compulsory we should know the size in advance, which may not possible always.

> Array declaration, creation & Initialization

a) Array Declaration

> At the time of declaration, we can't specify the size otherwise we will get compile time error.

E.g. int [] x; int [] [] x; int [][][] x;

b) Array Creation

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

E.g. int [] a = new int [3];

> At the time of array creation, compulsory we should specify the size otherwise we will get compile time error.

> It is legal to have an array with size zero in Java but negative size not allowed.

> At compile time, Compiler checks whether we're specifying some int value or not (only one thing). At runtime, JVM reserves the memory based on the given size.

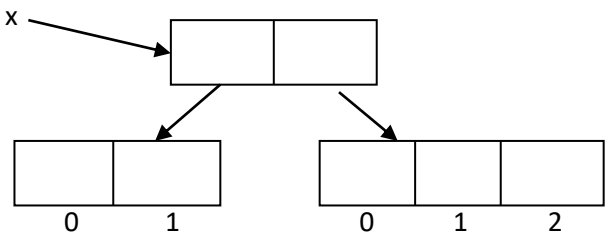
> To specify array size, the allowed datatypes are byte, short, int, char. So max allowed array size is $2^{31}- 1 = 2147483647$

> In Java, 2D array is not implemented by using matrix style instead implemented as array of arrays approach for multidimensional array creation.

For e.g. int [][] x = new int[2][]

x [0] = new int[2]

x [1] = new int[3]



c) Array Initialization

> Once we create an array, every array element is by default initialized with default values.

E.g. int[] x = new int[3]

System.out.println(x); // [I@6bc7c54 <- classname@hashcode_in_hexadecimal

System.out.println(x[0]); // 0

> Whenever we're trying to print any reference variable, internally toString() method of Object class will be called which is implemented by default to return the String in the following form

className@hashcode_In_Hexadecimalform

> If we're trying to perform any operation on null, then we will get NullPointerException.

> Index of array should be integer.

> If we're trying to access array element out of range index, then we will get runtime exception saying ArrayIndexOutofBoundException.

> We can declare, create & initialize an array in a single line.

E.g. int[] x = {10, 20, 30}; String[] s = {"A", "AA"}

Int[][] x = {{10, 20}, {30, 40, 50}};

> **length VS length()**

a) length:- length is a final variable applicable for arrays. It represents the size of the array. Not Applicable for String objects.

E.g. int[] x = new int[6];

System.out.println(x.length) // 6

> In multidimensional array, length variable represents only base size but not total size. For total size, find sum of each array in multidimensional array i.e. x[0].length + x[1].length + …. + x[n-1].length

E.g. int[][] x = new int[6][3]

System.out.println(x.length); // 6

System.out.println(x[0].length); //3

b) length ():- length() method is a final method applicable for String objects. It returns no. of characters present in the string. Not Applicable for arrays.

E.g. String s = "abc";

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

> Anonymous Array

- Sometimes we can declare an array without name, such type of nameless arrays is called Anonymous arrays.
- The main purpose of anonymous arrays is just for instant use (one time usage).
- We can create anonymous array as follows

```
new int[] {10, 20, 30, 40}
```

- While creating anonymous arrays, we can't specify the size otherwise we will get compile time error.
- We can create multi-dimensional anonymous arrays also

```
new int[][] {{10, 20}, {30, 40, 50}}
```

> Array elements assignment

No	Array Type	Allowed Element
1.	Primitive Arrays	Any type which can be implicitly promoted to declared type.
2.	Object type Arrays	Either declared type or its child class object.
3.	Abstract class type Array	Its child class objects.
4.	Interface type Array	Its implementation class objects are allowed.

> Array Variable assignments

a) **Case 1:**

> Element level promotions are not applicable at array level.

E.g. int[] x = {10, 20}; int[] b = x; // wrong

char[] ch = {'a', 'b'}; char[] c = c; // wrong

> But in the case of Object type arrays, child class type array can be promoted to parent class type array.

E.g. String[] s = {"A", "B"}; Object[] obj = s; // correct

b) **Case 2:**

> Whenever we're assigning one array to another array, internal elements won't be copied, just reference variable be reassigned.

c) **Case 3:**

> Whenever we're assigning one array to another array, both the dimensions & types must be matched but sizes are not required to match.

> Types of Variables

a) Division 1: - Based on the type of value represented by a variable, all variable are divided into 2 categories:-

1. **Primitive variable** – can be used to represent primitive values.
2. **Reference variable** – can be used to refer objects.

b) Division 2: - Based on position of declaration & behaviors, all variables are divided into 3 categories:-

1. Instance variable
2. Static variable
3. Local variable

1. **Instance variable (Object level variable or Attributes)**

> If the value of a variable is varied from Object to Object, such type of variables is called as Instance Variable.
> For every object, a separate copy of instance variables will be created.
> Instance variables should be declared within the class directly but outside of any method or block or constructor.
> Instance variables will be created at the time of Object creation & destroyed at the time of Object destruction; hence the scope of instance variable is exactly same as the scope of Object.
> Instance variables will be stored in the heap memory as the part of Object.
> We can't access instance variable directly from static area but we can access by using Object reference.
> We can access instance variable directly from instance area.
> For Instance variables, JVM will always provide default values & we're not required to perform initialization explicitly.
> Instance variables can be accessed by multiple threads simultaneously hence they are not thread-safe.

2. **Static variable (Class level variable or Fields)**

> If the value of a variable is not varied from Object to Object then it is not recommended to declare variable as instance variable. We have to declare such type of variable at class level by using “static” modifier.

> In the case of static variable, a single copy will be created at class level & shared by every object of the class.

> Static variables should be declared within the class directly but outside of any method or block or constructor.

> Static variables will be created at the time of class loading & destroyed at the time of class unloading. Hence scope of static variable is exactly same as scope of .class file.

> Static variables will be stored in Method area.

> We can access static variables either by object reference or by class name but recommended to use class name. Within the same class, it is not required to use class name & we can access directly.

> We can access static variables directly from both instance & static areas.

> For static variables, JVM will provide default values, & we’re not required to perform initialization explicitly.

> Static variables can be accessed by multiple threads simultaneously hence they are not thread-safe.

3. **Local variable (Temporary/Stack/Automatic variable)**

> Sometimes to meet temporary requirements of the programmers, we can declare variable inside a method or block or constructor, such types of variables are called local variable.

> Local variables will be stored inside stack memory.

> Local variables will be created while executing the block in which we declared it. Once block execution completes, automatically local variable will be destroyed. Hence the scope of local variable is the block in which we declared it.

>For local variables, JVM won’t provide default values, compulsory we should perform initialization explicitly before using that variable i.e. if we’re not using, then it is not required to initialize.

> The only applicable modifier for local variable is final.

> In case of local variables, for every thread a separate copy will be created & hence local variables are thread-safe.

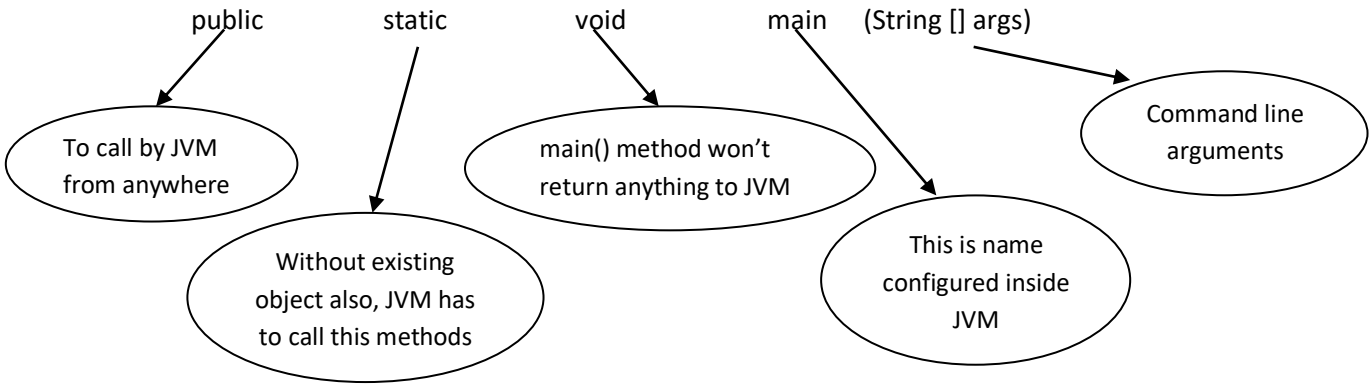
> Var-arg methods (variable no. of argument methods)

- We can declare a method, which can take variable no. of arguments such type of methods are called var-arg methods.
- We can declare a var-arg method as follows: methodName (int… x)
- We can call var-arg method by passing any no. of int values including zero number.
- Internally var-arg parameter will be converted into 1D array, hence within the var-arg method, we can differentiate values by using index.
- We can mix var-arg parameter with normal parameters but then var-arg parameter should be last parameter.

  E.g. m1 (char ch, String… s);
- Inside var-arg method, we can take only one var-arg parameter & we can’t take more than one var-arg parameter.
- Inside a class, we can’t declare var-arg method & the corresponding 1D array method simultaneously.
- In General, var-arg method will get least priority i.e. if no other method matched then only var-arg method will get a chance. It is exactly same as default case inside switch.
- Whenever 1D array present, we can replace it with var-arg parameter but we can’t replace var-arg parameter present with 1D array.

> Main Method

- At runtime, JVM always searches for the main () method with the following prototype:



> The above syntax is very strict & if we perform any change then we will get Runtime Exception "NoSuchMethodError: main".

> We can perform following changes:

a) Instead of public static we can take static public i.e. the order of modifiers is not important.

b) We can replace String [] with var-arg parameter.

c) Instead of args, we can take any valid java identifier.

d) We can declare main() method with following modifiers : final, synchronized, strictfp

i.e. final synchronized strictfp public static void main(String… args) is valid declaration.

> **Case 1:** Overloading of the main () method is possible but JVM will always call String [] argument main () method only. The other overloaded method, we have to call explicitly like normal method call.

Case 2: Inheritance concept applicable for main () method by executing child class, if child doesn't contain main() method then parent class main method will be executed.

Case 3: It seems overriding concept applicable for main () method but it's not overriding but Method Hiding.

Note:

> From 1.7 version onward, main () method is mandatory to start program execution hence even though class contains static block, it won't be executed if the class doesn't contain main () method.

> Command Line Arguments

- The arguments which are passing from command prompt are called command line argument.
- With these command line arguments, JVM will create an array & by passing that array as argument JVM will call main () method.
- The main objective of command line argument is we can customize behavior of the main () method.
- Usually space itself is the separator between command line arguments. If our command line arguments itself contains space then we have to enclose that command line argument with "";

> Java Coding Standards

- Usually class names are Nouns. Should start with uppercase character & if it contains multiple words every inner word should start with uppercase character. For e.g. StringBuffer

- Usually interface names are adjectives & should start with uppercase character & if it contains multiple words, every inner words starts with uppercase character. For e.g. Runnable

- Usually method names are either verb or verb & noun combination & should start with lowercase alphabet symbol & if it contains multiple words then every inner word should start with uppercase character (Camel Case convention).
For e.g. print() , getName()

- Usually variable names are nouns & should start with lowercase alphabet symbol & if it contains multiple words then every inner word should start with uppercase character. (Camel Case convention) For e.g. name, mobileNumber

- Usually constant names are nouns & should contain only uppercase characters & if it contains multiple words then these words are separated with "_" symbol. For e.g. MAX_VALUE, MAX_PRIORITY

- Usually we can declare constants with public, static, final modifiers.

- Java Bean is a simple Java class with private properties & public getter & setter methods.