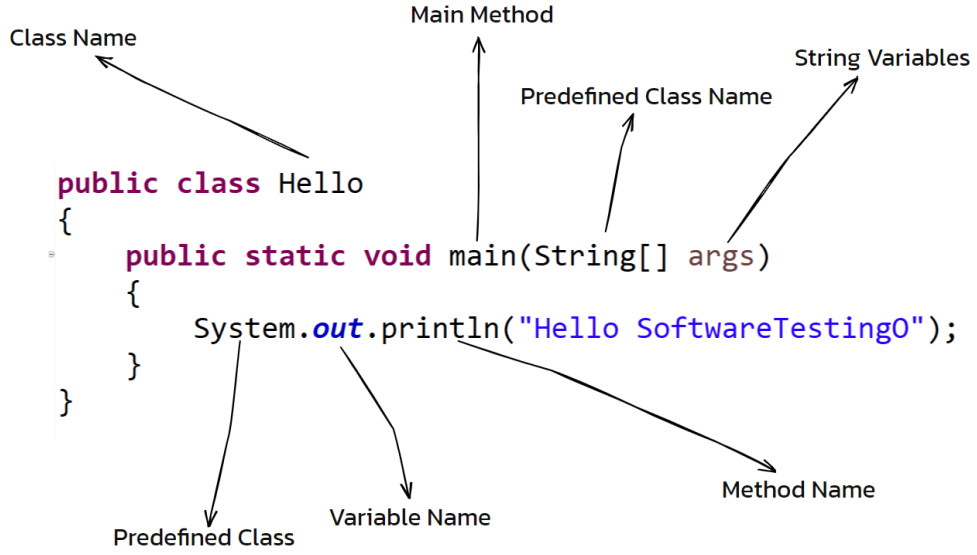
Java Language Fundamentals

Identifiers:

Name in a java program is called an identifier.
Name can be

- Class name
- Method name
- Variable name



Rules for identifiers:

Java Identifiers

All Java components require names. Names used for classes, variables, and methods are called **identifiers**.

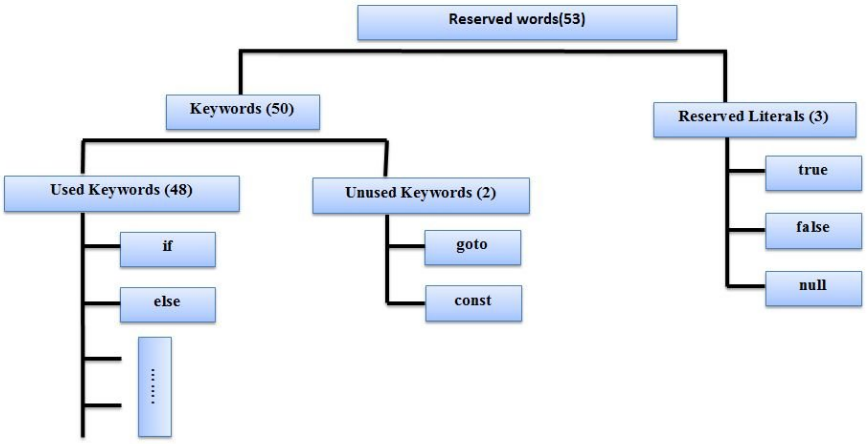
In Java, there are several points to remember about identifiers. They are as follows –

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- A key word cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of legal identifiers: age, $salary, _value, __1_value.
- Examples of illegal identifiers: 123abc, -salary.

Identify if the below are valid identifiers:

1)_$_ (valid)
2)Ca$h (valid)
3)Java2share (valid)
4)all@hands (invalid)
5)123abc (invalid)
6)Total# (invalid)
7)Int (valid)
8)Integer (valid)
9)int (invalid)
10) tot123

Reserved words:
Identifiers that are already reserved are called reserved words. Below are the reserved words:

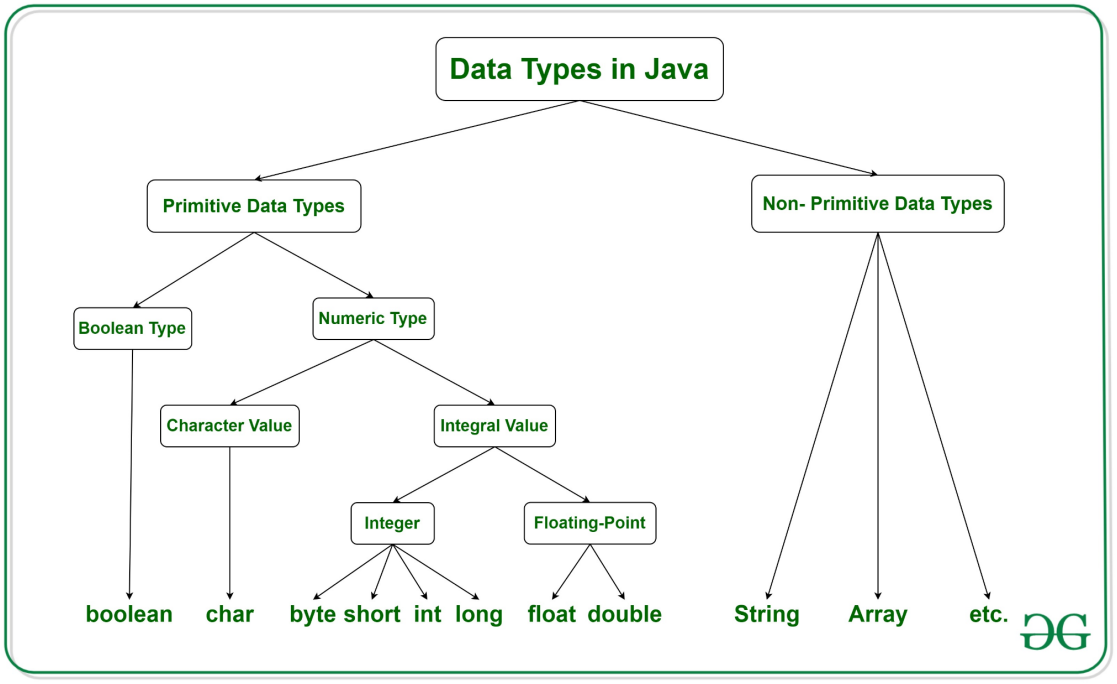


Java Keywords				
abstract	assert	boolean	break	byte
case	catch	char	class	continue
default	do	double	else	enum
extends	final	finally	float	for
if	implements	import	instanceof	int
interface	long	native	new	package
private	protected	public	return	short
static	strictfp	super	switch	synchronized
this	throw	throws	transient	try
void	volatile	while		
Keywords that are not currently used				
const	goto			

Java Data Types:

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.



Details of primitive data types in java:

DATA TYPES	SIZE	DEFAULT	EXPLAINATION
boolean	1 bit	false	Stores true or false values
byte	1 byte/ 8bits	0	Stores whole numbers from -128 to 127
short	2 bytes/ 16bits	0	Stores whole numbers from -32,768 to 32,767
int	4 bytes/ 32bits	0	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes/ 64bits	0L	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes/ 32bits	0.0f	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes/ 64bits	0.0d	Stores fractional numbers. Sufficient for storing 15 decimal digits
char	2 bytes/ 16bits	'\u0000'	Stores a single character/letter or ASCII values

Java escape characters:

Escape Sequences

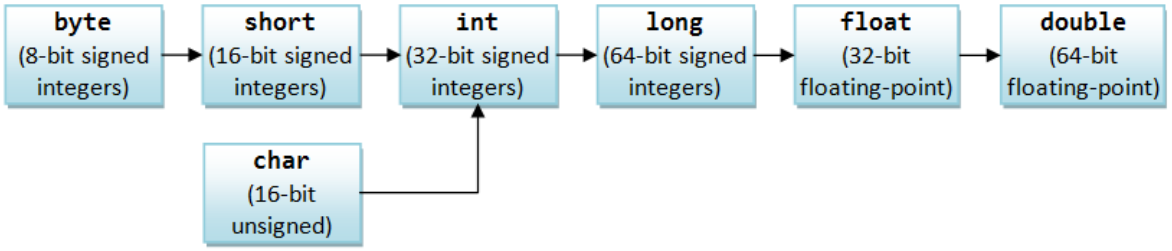
Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

```
public class EscapeCharacterExample {
   public static void main(String args[]) {
        //inserts a Tab Space
        String str = "James\tGosling";
        System.out.println(str);
        //inserts a New Line
        String str1 = "Push yourself!\nNobody is going to do that for you!";
        System.out.println(str1);
        //it inserts a backslash
        String str2 = "And\\Or";
        System.out.println(str2);
        //it inserts a Carriage. Brings cursor to start of line.
        String str3 = "Hello healthy \rWorld";
        System.out.println(str3);
        //it prints a single quote
        String str4 = "Oracle\'s Java";
        System.out.println(str4);
        //it prints double quote
        String str5 = "New\'Twilight'Line";
        System.out.println(str5);
   }
}
```

Output:

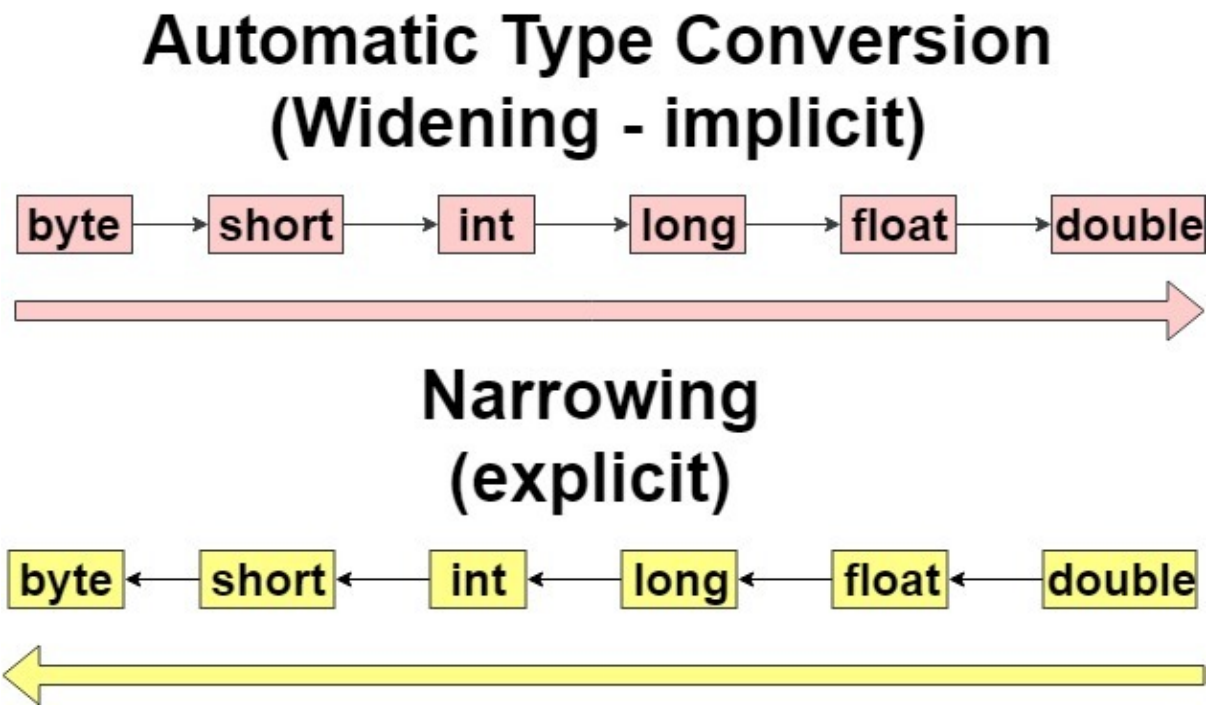
```
James  Gosling
Push yourself!
Nobody is going to do that for you!
And\Or
World
Oracle's Java
New'Twilight'Line
```

Primitive data types type casting:



Orders of Implicit Type-Casting for Primitives

Types of type casting:




```
class ImplicitTypeCasting {
   public static void main(String[] args) {
        int i = 100;

        //automatic(implicit) type conversion
        long l = i;

        //automatic(implicit) type conversion
        float f = l;
        System.out.println("Int value " + i);
        System.out.println("Long value " + l);
        System.out.println("Float value " + f);
   }
}
```

Output:

Int value 100
Long value 100
Float value 100.0

```
class ExplicitTypeCasting {
   public static void main(String[] args) {
        double d = 100.04;

        //explicit type casting(Narrowing)
        long l = (long) d;

        //explicit type casting(Narrowing)
        int i = (int) l;
        System.out.println("Double value " + d);

        //fractional part lost
        System.out.println("Long value " + l);

        //fractional part lost
        System.out.println("Int value " + i);
   }
}
```

Output:

Double value 100.04
Long value 100
Int value 100

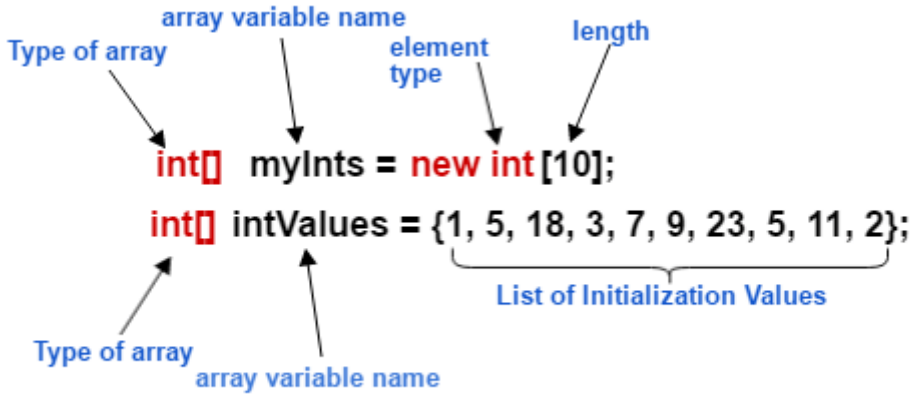
Arrays in Java:

Arrays in Java are non-primitive data types that store elements of a similar data type in the memory. Arrays in Java can store both primitive and non-primitive types of data in it.

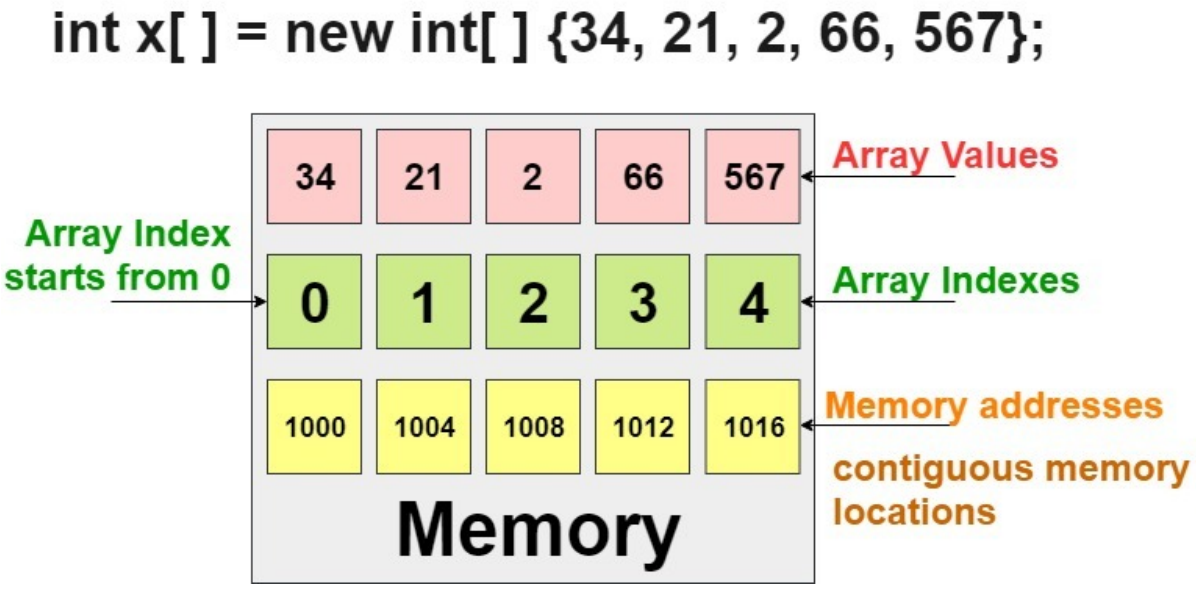
There are two types of arrays, single-dimensional arrays have only one dimension, while multidimensional arrays have 2D, 3D, and nD dimensions.

```
public class ArraysExample {
   public static void main(String[] args) {
        int[] myInts = new int[10];
        int[] intValues = {1, 5, 18, 3, 7, 9, 23, 5, 11, 2};

        System.out.println(myInts.length);
        System.out.println(intValues.length);
   }
}
```



Memory consumption:



Total Memory consumption: 5 * 4 bytes = 20 bytes.

Arrays Basic operations:

```
public class ArraysBasicOperations {
   public static void main(String[] args) {
       int[] inputArray = {1, 5, 18, 3, 7, 9, 23, 5, 11, 2};
       // Find array length/size
       System.out.println(inputArray.length);
       // Find element at index 5
       System.out.println(inputArray[5]);
       // Find last element in array
       System.out.println(inputArray[inputArray.length - 1]);
```

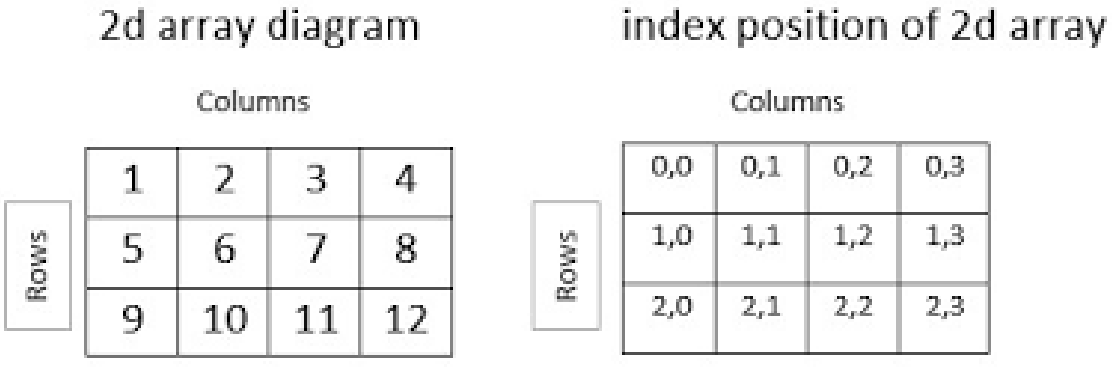
```
        // Print elements in array (Wrong way)
        System.out.println(inputArray);
        // Print elements in array
        for (int i = 0; i < inputArray.length; i++) {
            System.out.print(i + " ");
        }
        // Print elements in array enhanced for loop
        System.out.println("\nUsing enhanced for loop");
        for (int i : inputArray) {
            System.out.print(i + " ");
        }
    }
}
```

Output:

```
10
9
2
[I@6acbcfc0
0 1 2 3 4 5 6 7 8 9
Using enhanced for loop
1 5 18 3 7 9 23 5 11 2
```

Two Dimensional Arrays:

```
public class Arrays2D {
    public static void main(String[] args) {
        int[][] twoDimensionArray = {
                {1, 2, 3, 4},
                {5, 6, 7, 8},
                {9, 10, 11, 12}
        };
    }
}
```



Basic operations:

```
public class Arrays2DBasicOperations {
   public static void main(String[] args) {
       int[][] twoDimensionArray = {
               {1, 2, 3, 4},
               {5, 6, 7, 8},
               {9, 10, 11, 12}
       };
       //Always number of rows length
       System.out.println(twoDimensionArray.length);
       //first row length
       System.out.println(twoDimensionArray[0].length);
       //second row length
       System.out.println(twoDimensionArray[1].length);
       //third row length
       System.out.println(twoDimensionArray[2].length);
       // Find last element in second row
       System.out.println(twoDimensionArray[1][3]);
       // Print elements in array (Wrong way)
       System.out.println(twoDimensionArray);
       // Print elements in array (Right way)
       for (int i = 0; i < twoDimensionArray.length; i++) {
           for (int j = 0; j < twoDimensionArray[i].length; j++) {
               System.out.print(twoDimensionArray[i][j] + " ");
           }
           System.out.println();
       }
   }
}
```

Output:

```
3
4
4
4
8
[[I@6acbcfc0
1 2 3 4
5 6 7 8
9 10 11 12
```

.

Note:

Java supports arrays of both primitive and Object types:

```
public class ArraysExampleString {
    public static void main(String[] args) {
        String[] trafficSignalColors = {"Red", "Yellow", "Green"};
    }
}
```

Main method:

The main() is the starting point for JVM to start execution of a Java program. Without the main() method, JVM will not execute the program. The syntax of the main() method is:

```
public static void main(String[] args){
     //Code here…
}
```

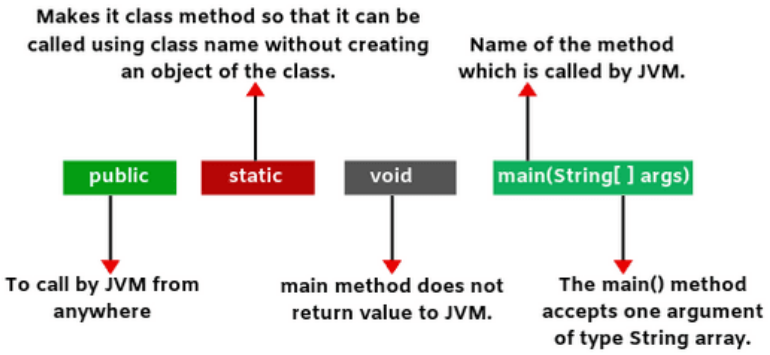


Fig: Java main method

Understanding the Main Method

The main method serves as the starting point for the execution of a Java program. When a Java application is run, the Java Virtual Machine (JVM) looks for the main method and begins executing the code inside it. Let's break down its syntax to grasp its components:

1. public: The main method is typically declared as public, allowing it to be accessible from anywhere within the program.
2. static: The main method is declared as static, meaning it belongs to the class itself rather than an instance of the class. This allows the JVM to call the main method without having to create an object of the class.
3. void: The main method does not return any value. It is responsible for executing the code within it rather than producing a result.
4. main: The name "main" is fixed and recognized by the JVM as the entry point for program execution. It must be spelled exactly as "main."
5. String[] args: The main method accepts a parameter called "args," which is an array of strings. This parameter allows the passing of command-line arguments to the program during execution.

Java Naming Convention:

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

Code Listing 48: Naming Convections in Java

Avoid

```
1.  class icecream{
2.   int flavourtype;
3.   final int size=2;
4.   void
     getflavourtype(){
5.    return
     flavourtype;
6.   }
7.  }
```

Preferable

```
1.  class IceCream{
2.    int flavourType;
3.    final int SIZE=2;
4.    void
     getFlavourType(){
5.      return
     flavourType;
6.    }
7.  }
```