



Java Introduction

Hexavarsity



Objective

- Introduction to Java
- Java Editions
- Features of Java
- Datatype
- Operator
- Variables
- Control statements
- Array





Introduction to Java

History of Java

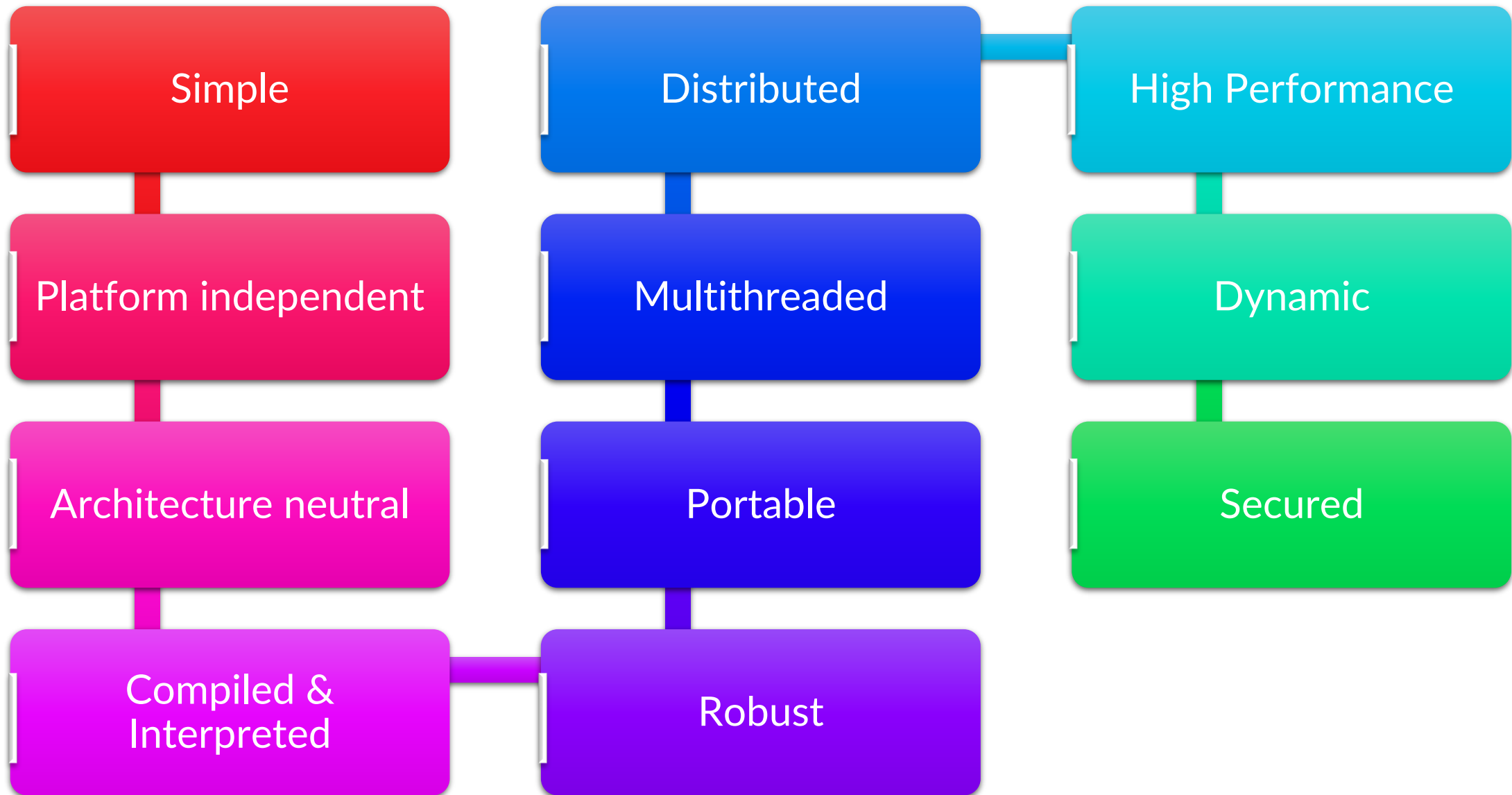


- Java - developed by Sun Microsystems in 1991. now owned by Oracle
- Java Authors: James Gosling
- JDK 1.0 is released on (January 23, 1996)
- Current version of java is Java 18 (March, 2022)
- Desktop Applications, Enterprise and Web Applications, Mobile Application, Robotics and Games, Automation testing are the area java is used



- **Java SE (Java Standard Edition)**
 - It includes Java programming APIs such as import packages, OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection etc.
- **Java EE (Java Enterprise Edition)**
 - Mainly used to develop web and enterprise applications It includes topics like Servlet, JSP, Web Services, EJB, etc.
- **Java ME (Java Micro Edition)**
 - Used to develop Setup box, handheld devices

Features of Java



Simple

- Syntax is based on C & C++.
- Removed pointers, operator overloading etc.
- Automatic Garbage Collection.

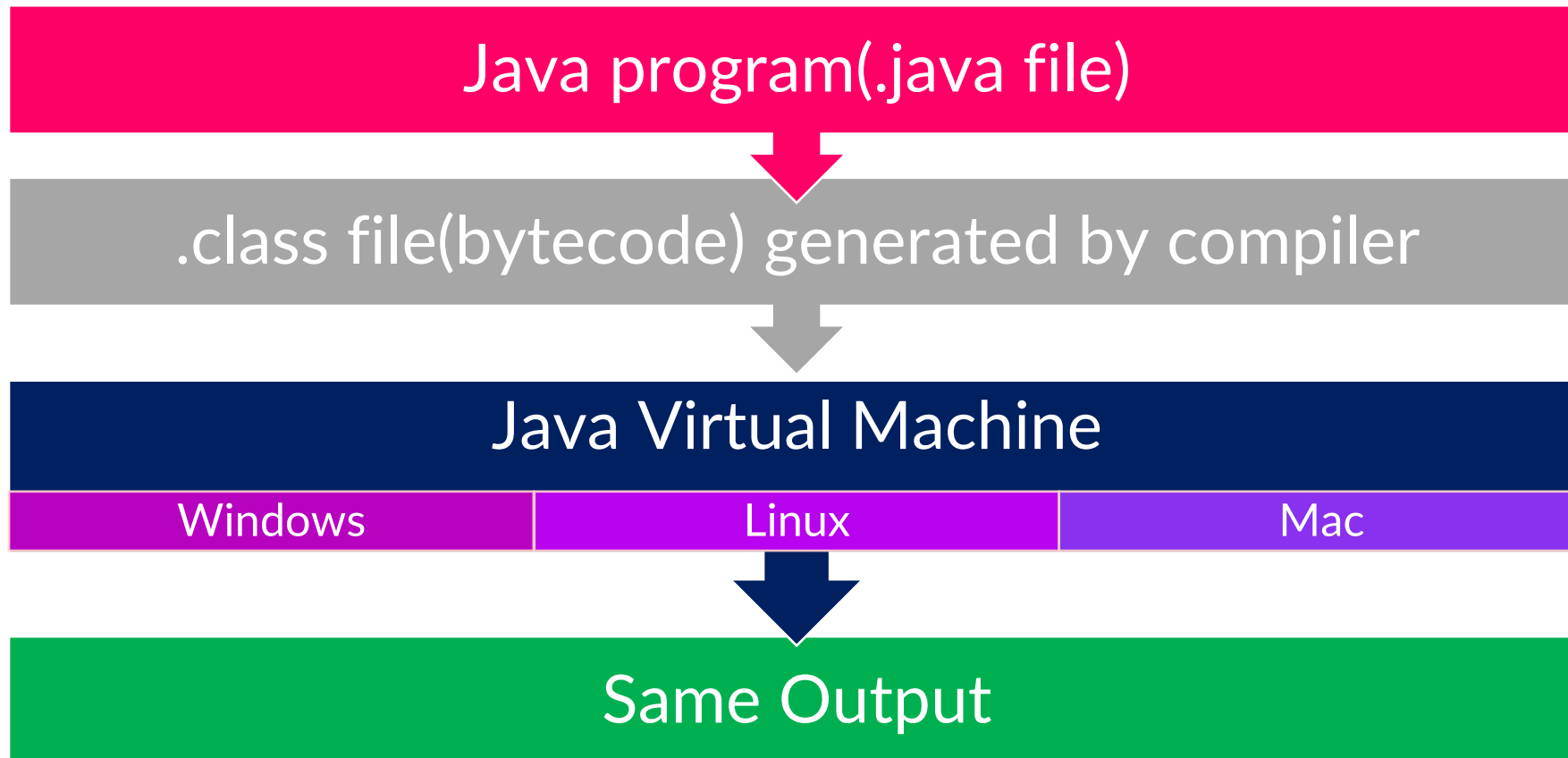
Portable

- We can carry the java code that written in one platform to any platform.(Windows, Linux, Mac etc.)



Platform independent

- Write Once Run Anywhere (WORA).
- Executing java program is independent of the Operating System of the System.



PROCESSOR UPDATING



- C Source file need changes
- Primitive Data type size may be different for compiler to compiler

OS UPDATING



- Java Source file no need any changes
- Primitive Data type size are same

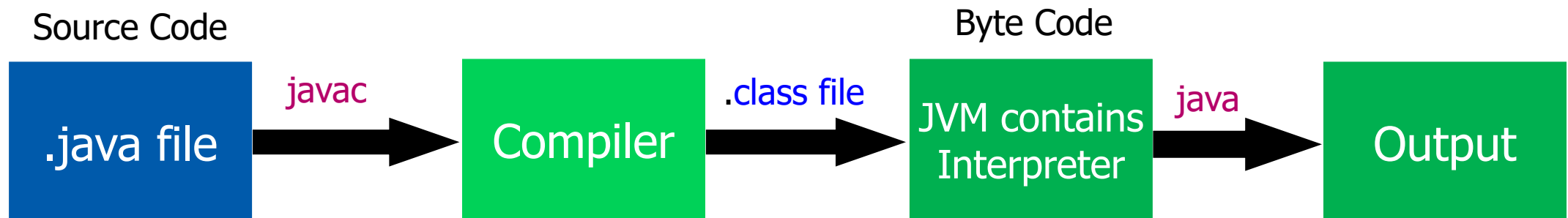
Robust

- Robust means strong. Java program doesn't crash easily at runtime.
- Strongly typed
- Inbuilt exception handling.
- Automatic Garbage collection

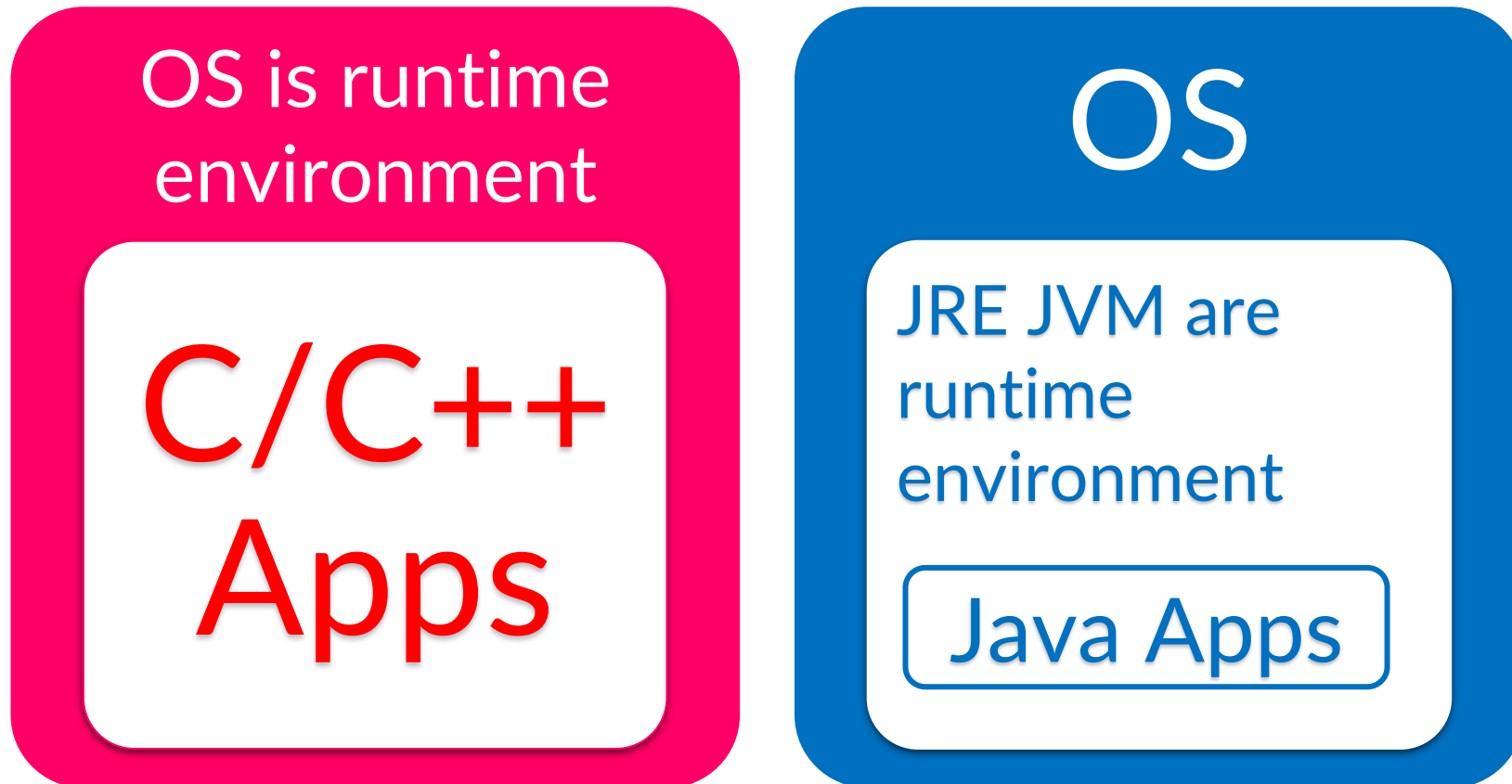


Java is Compiled and Interpreted

- Java compiler converts Java Source code into Java bytecode.
- Java interpreter converts bytecode into machine code that can be directly executed by the machine.

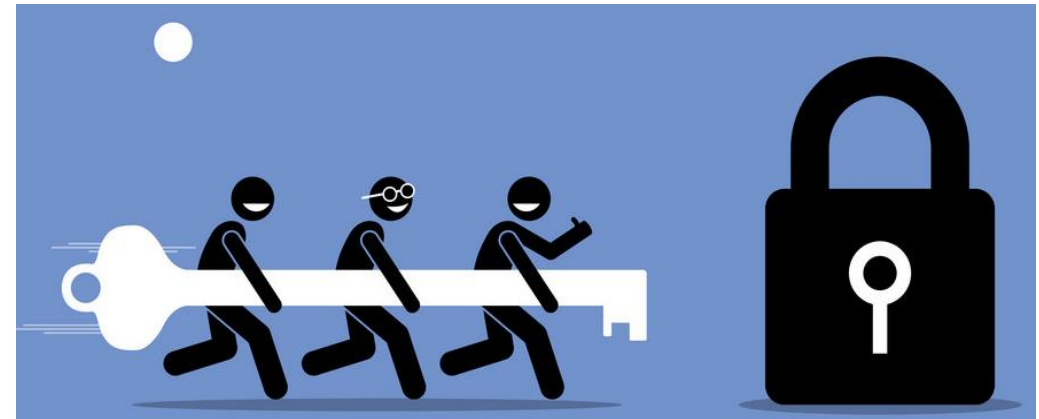


- Security problem like eavesdropping, tampering, impersonation and virus threats can be minimized using java on internet.



Multi-threaded

- Java programs that deal with multiple tasks at same time. Multithreading achieved with help of Thread class and interface.
- The same memory and resources are used by multiple threads running at the same time

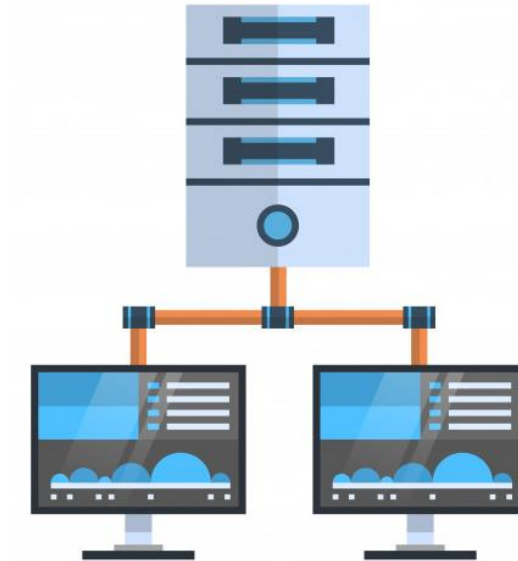


High Performance

- Runtime execution performance are improve by the help of JIT compiler a part of JVM. Compared to C and C++ java is bit slow.

Distributed

- We can create distributed applications in java RMI and EJB are used for creating distributed applications.
- We may access files by calling the methods from any machine on the internet.



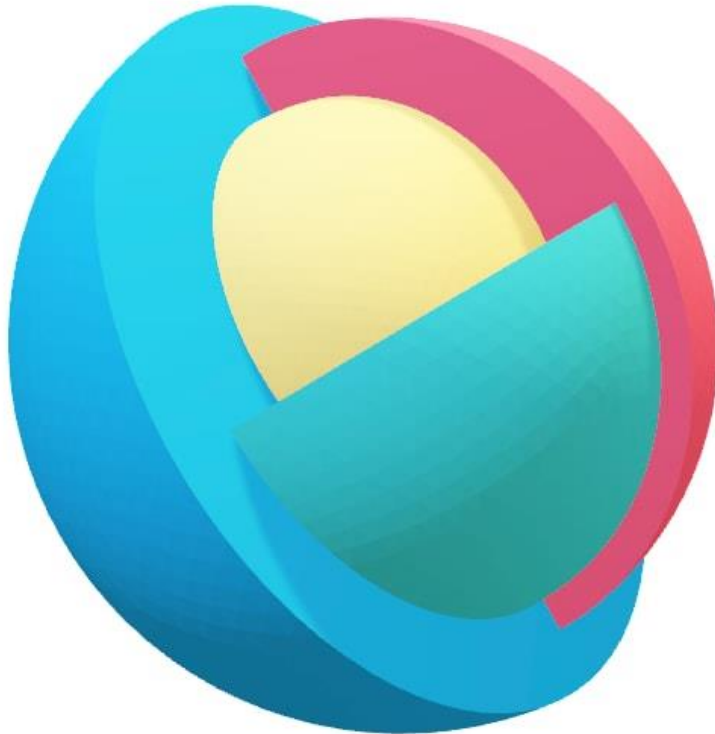
Dynamic

- java class are loaded at runtime on demand



JVM Architecture





JVM

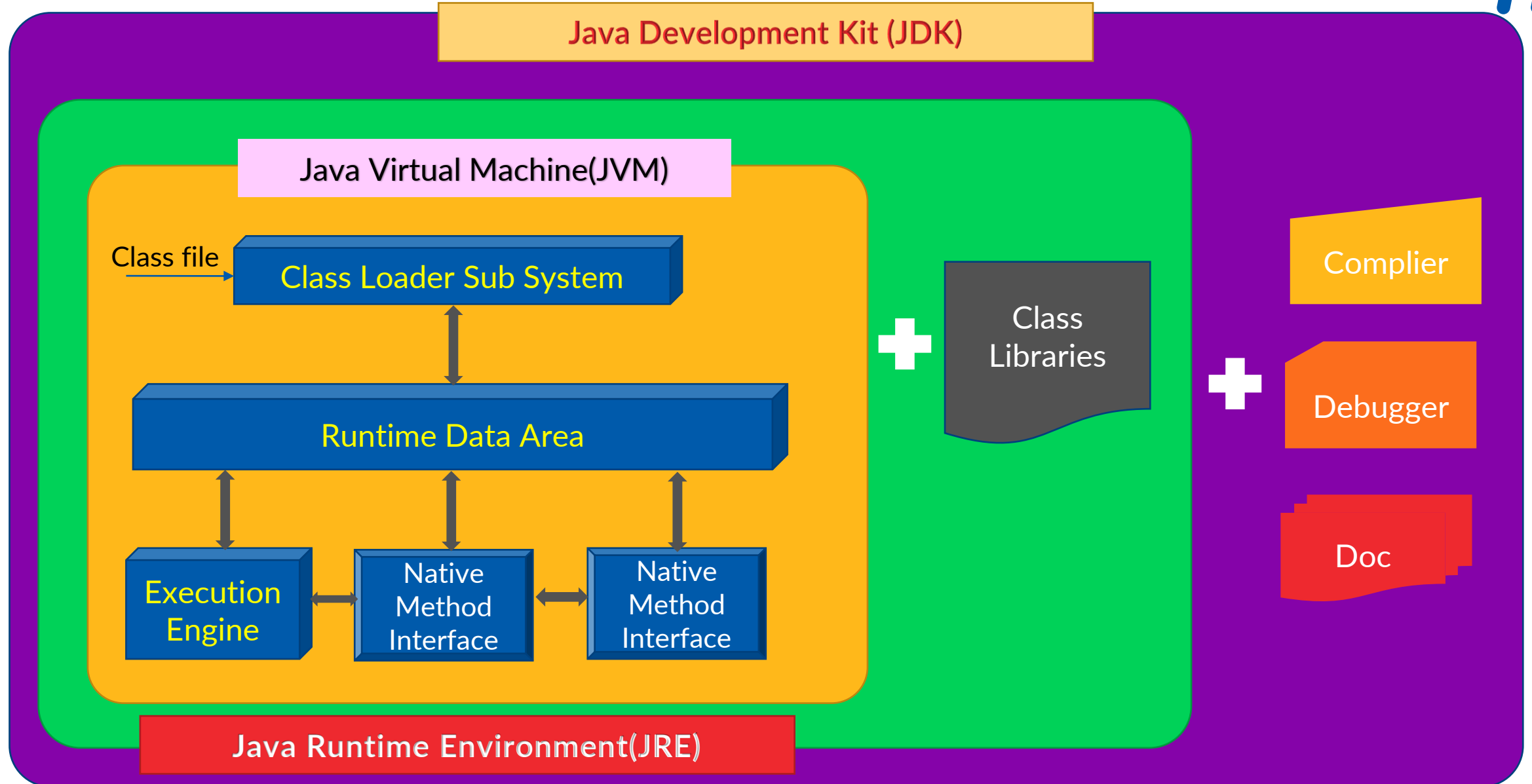
Converts bytecode to machine-specific code.

JRE

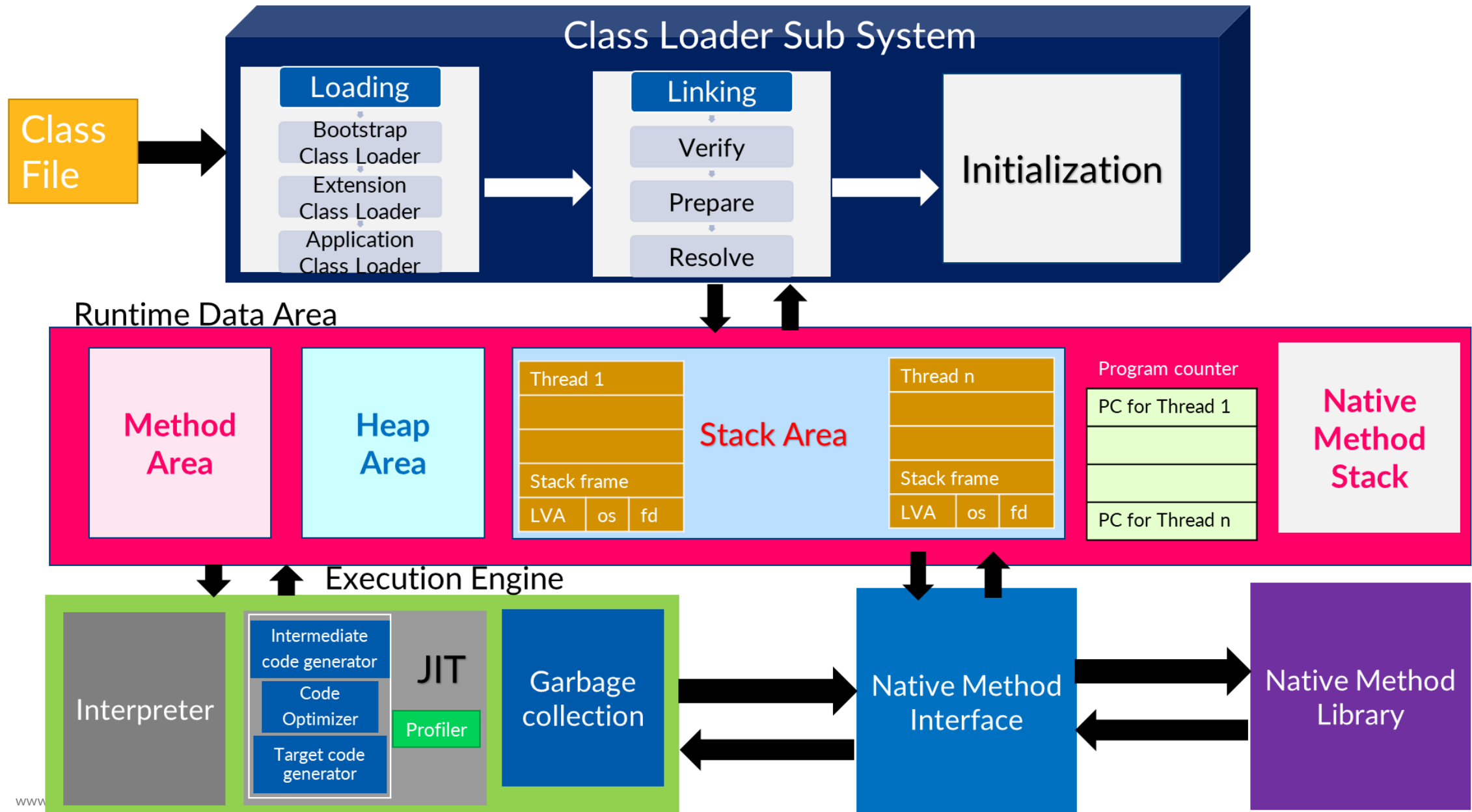
Executes Java programs.
Includes JVM.

JDK

The development kit that includes JRE, a Java compiler, a debugger and more.



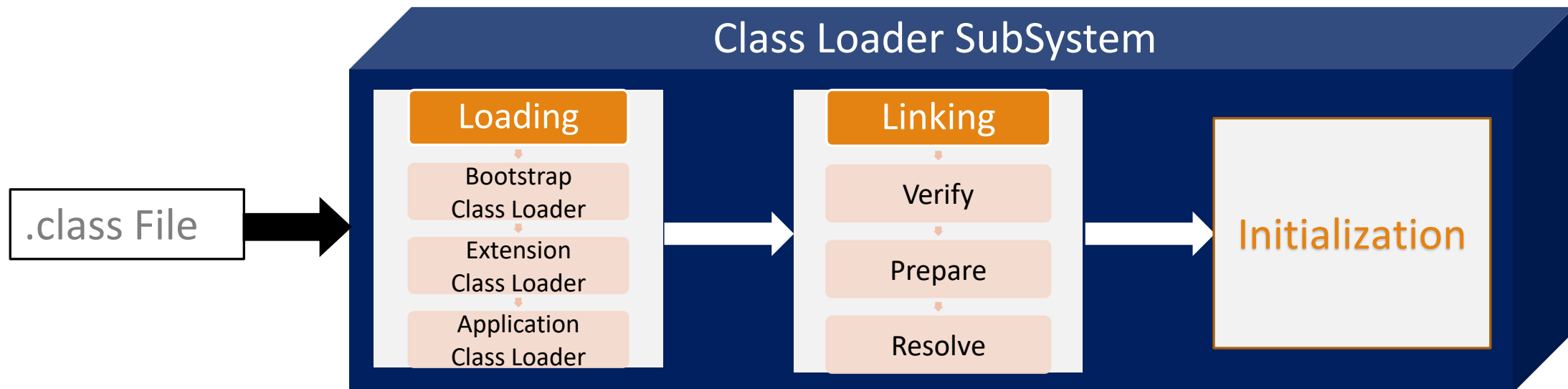
JVM Architecture



Class Loader Subsystem



- Class Loader Subsystem will perform three activities.
 1. Loading
 2. Linking
 3. Initialization





- The Class loader reads the “.class” file and save its binary data in the method area.
- For each “.class” file, JVM stores
 - ✓ Fully qualified name of the loaded class and its immediate parent class
 - ✓ AccessModifier, Variables and Method, constructor, scp information etc.
- JVM creates an object for these classes and store in the heap memory.
- These class level information like the name of the class, parent name, methods and variable information etc. can be retrived by getClass() method of Object class.

Linking

- Linking Performs verification, preparation, and resolution.
- **Verification:** Verify the bytecode in class file whether it is properly formatted and generated by a valid compiler or not. If verification fails, we get run-time exception `java.lang.VerifyError`.
- **Preparation:** JVM allocates memory for class variables and initialize the default values.

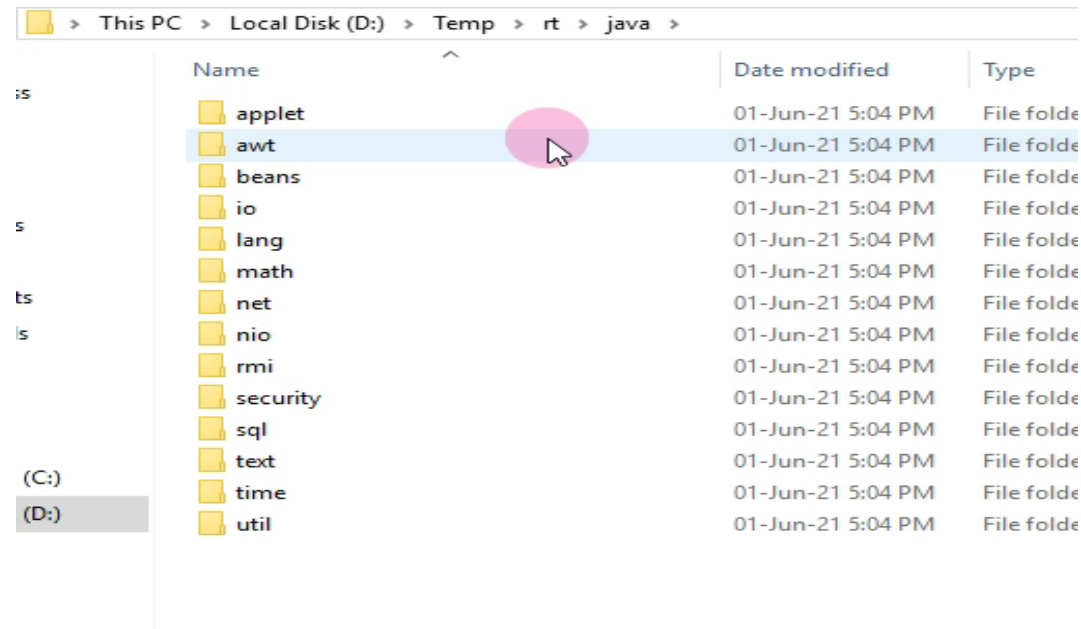
Initialization

- if any static block and static variables are assigned with their values defined in the code and.
- This is executed from top to bottom in a class and from parent to child in the class hierarchy.

Class loader types



- **Bootstrap class loader:** bootstrap class loader, capable of loading core java API classes present in the “JAVA_HOME/jre/lib” directory.



- **Extension class loader:** It loads the classes present in the extensions directories “JAVA_HOME/jre/lib/ext”.
- **System/Application class loader:** It load classes from the application classpath. It uses Environment Variable which linked to java.class.path.



- **Method area:**
 - ✓ It is a memory block to store class level information code like (class name, immediate parent class name, variable, methods) and static variable
- **Heap:**
 - ✓ It is a memory block to store objects created by JVM
- **Java Stack:**
 - ✓ For every thread, JVM creates one run-time stack, each stack frame stores methods calls. All local variables of that method after completing the execution it will be destroyed by JVM
- **Program Counter:**
 - ✓ Store address of current execution instruction of a thread. Obviously, each thread has separate PC Registers.
- **Native method stacks:**
 - ✓ For every thread, a separate native stack is created. It stores native method information.

Execution Engine

- It reads the byte-code line by line, executes the instructions if there any illegal operation like accessing array element beyond the size will generate.
- **Interpreter:** It interprets the bytecode line by line and then executes. when one method is called multiple times, every time interpretation, translating code is required.

Just-In-Time Compiler(JIT)

- It is used to increase the performance of an interpreter. whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not required, thus efficiency is improved.

Garbage Collector

- It destroys un-referenced objects.



Java Native Interface (JNI) :

- It is an interface that interacts with the Native Method Libraries and provides the native libraries(C, C++) required for the execution

Native Method Libraries

- It is a collection of the Native Libraries(C, C++) which are required by the Execution Engine.



Keywords

Keywords



• Datatype

Control
Statements

Access Modifiers

Exception
Handling

class

Unused

Others

Literals

returnType

Object

53 Keywords



<u>Datatype</u>	<u>Control Statements</u>	<u>Access Modifiers</u>	<u>Exception Handling</u>	<u>class</u>	<u>Unused</u>
1. byte	1. if	1. public	1. try	1. package	1. goto
2. short	2. else	2. private	2. catch	2. class	2. const
3. int	3. do	3. protected	3. finally	3. interface	<u>Others</u>
4. long	4. while	4. static	4. throw	4. import	
5. float	5. for	5. final	5. throws	5. extends	1. enum
6. double	6. switch	6. abstract	6. assert	6. implements	
7. boolean	7. case	7. synchronized		<u>Object</u>	
8. char	8. break	8. transient	<u>returnType</u>	1. new	
<u>Literals</u>	9. default	9. volatile	1. void	2. instanceof	
1. true	10. continue	10. native		3. this	
2. false	11. return	11. strictfp		4. super	
3. null					



Naming Conventions

Naming Conventions



- Naming conventions make programs more understandable by making them easier to read.
- They can also give information about the function of the identifier-for example, whether it's a constant, package, or class-which can clarify the code.

Identifier Type	Rules for Naming	Examples
Packages	<p>The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org,</p> <p>Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.</p>	<p>com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese</p>
Classes	<p>Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive.</p>	<p>class Raster; class ImageSprite;</p>

Naming Conventions



Identifier Type	Rules for Naming	Examples
Interfaces	Interface names should be capitalized like class names.	<code>interface RasterDelegate; interface Storing;</code>
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	<code>run(); runFast(); getBackground();</code>
Variables	<ul style="list-style-type: none">• Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters.• Variable names should not start with underscore <code>_</code> or dollar sign <code>\$</code> characters, even though both are allowed.• Variable names should be short yet meaningful.• One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are <code>i</code>, <code>j</code>, <code>k</code>, <code>m</code>.	<code>int i; char c; float myWidth;</code>
Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores (" <code>_</code> ").	<code>static final int MIN_WIDTH = 4;</code>

Find the identifiers



```
class Sample
{
    public static void main(String[] args)
    {
        int age=22;
    }
}
```

Identifiers are the names of variables, methods, classes, packages and interfaces.

1. Sample
2. main
3. String
4. args
5. age



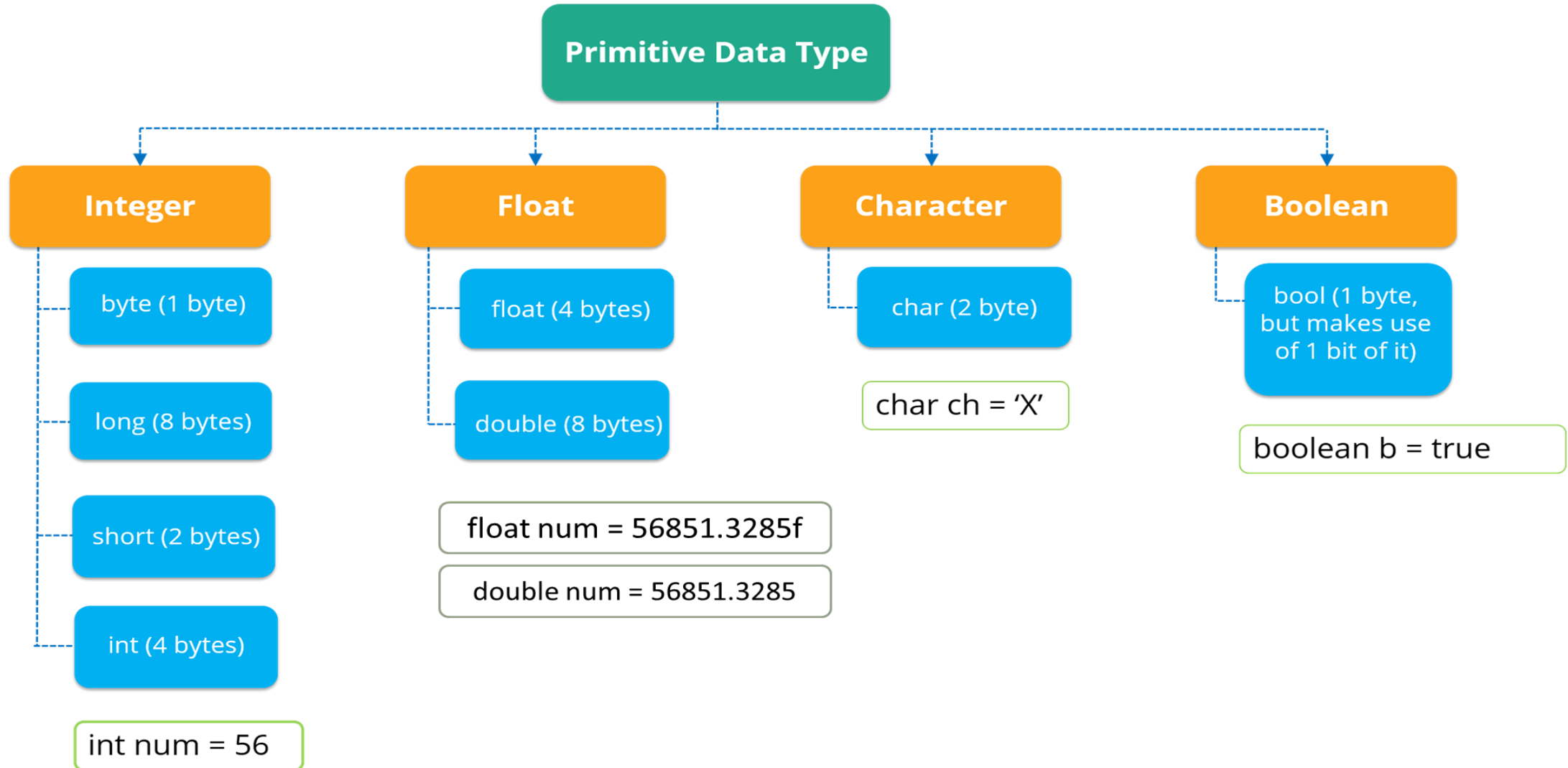
Data Type



What is Data type?



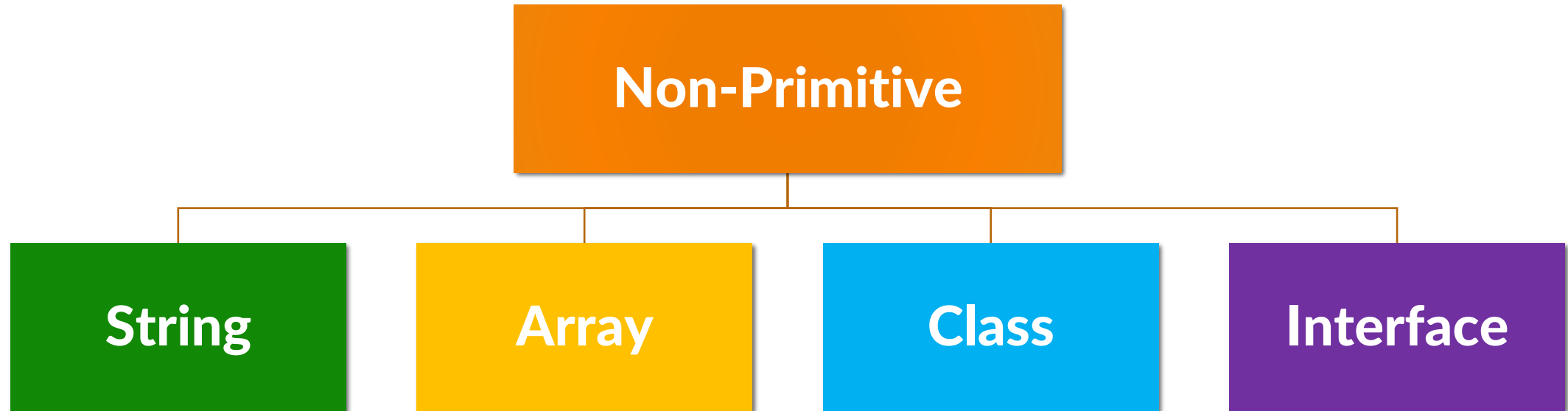
- Represent different values which are stored in a variable. Primitive and non-primitive are different datatypes.



Non-Primitive Datatypes



- Non-Primitive data types refer to objects and hence they are called reference types.



Wrapper Class?



- Wrapper class provides a mechanism to convert primitive data types into wrapper class objects.

Primitive Data Type	Wrapper Class	Methods
int	Integer	intValue()
char	Character	charValue()
float	Float	floatValue()
boolean	Boolean	booleanValue()
double	Double	doubleValue()
short	Short	shortValue()
long	Long	longValue()
byte	Byte	byteValue()



Need Of Java Wrapper Class

- They convert the primitive data types into objects.
- The classes in java.util package only works with objects.
- Data structures in the collection framework only store objects.

```
class wrapperClass{  
    public static void main(String args[]){  
  
        Integer myInt = 10;  
        Character myChar = "C2C";  
        Float myFloat = 10.25;  
        System.out.println(myInt.intValue());  
        System.out.println(myChar.charValue());  
        System.out.println(myFloat.floatValue());  
    }  
}
```

Example:

Output :
10
C2C
10.25

Autoboxing



- Autoboxing is the automatic conversion of the primitive data types into objects of their corresponding wrapper class.

```
class BoxingExample1{
    public static void main(String args[]){
        int a=50;
        Integer a2=new Integer(a);//Boxing

        Integer a3=5;//Boxing

        System.out.println(a2+" "+a3); //50 5
    }
}
```

Example:

```
import java.util.ArrayList;
class Autoboxing {
    public static void main(String args[]){
        char ch = 'e';
        Character e = ch;
        ArrayList<Integer> arraylist = new ArrayList<Integer>();
        arraylist.add(10); //Boxing
        System.out.println(arraylist.get(0)); // 10
    }
}
```

Example:

Unboxing



- It is the reverse of autoboxing, where the wrapper class object is converted to their corresponding primitive data type.

```
class UnboxingExample1{  
    public static void main(String args[]){  
        Integer i=new Integer(50);  
        int a=i; // Unboxing  
  
        System.out.println(a); //50  
    }  
}
```

Example:

```
import java.util.ArrayList;  
class Unboxing{  
    public static void main(String args[])  
    {  
        Character ch = 'e';  
        char 'e' = ch;  
  
        ArrayList<Integer> arraylist = new ArrayList<Integer> ();  
        arraylist.add(10);  
        int number = arraylist.get(0); // Unboxing  
        System.out.println(number); //10  
    }  
}
```

Example:

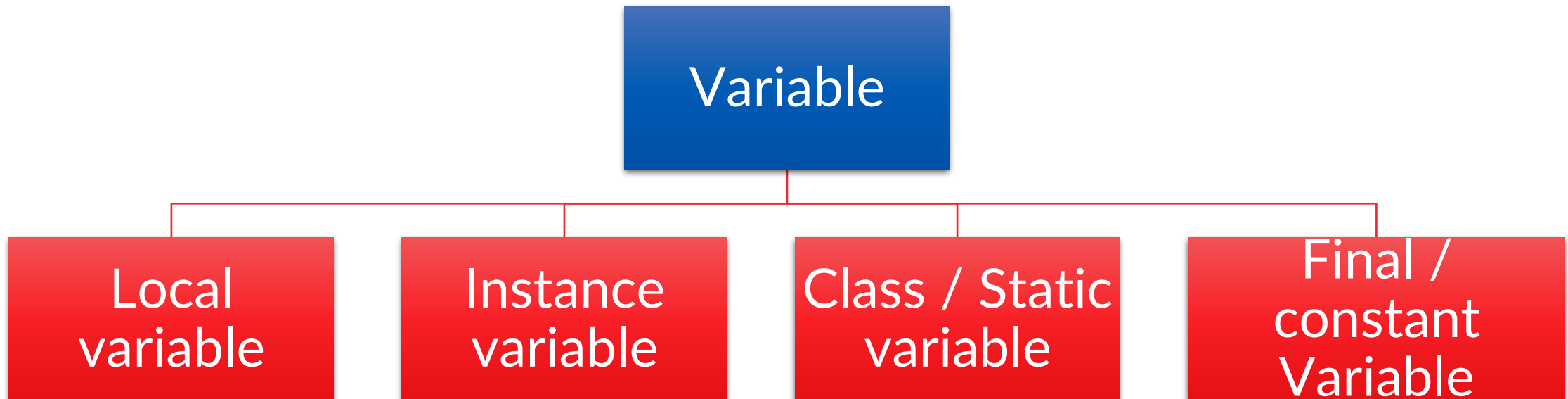


Variables

Variables



- It is used to store a data value.
- These variables are members of a class.





Local variable

- These are the variables which are declared within the method of a class.
- Should not declared with access specifier
- Variable cannot access outside of method

Instance variable:

- Instance variable is declared in a class.
- Can be declared with access specifier (private, public, protected)
- Variable can be accessed in method using object.

Class variable

- Class variables are also called as static variables.
- Class variable declared in class scope with keyword static.
- These variables have only one copy that is shared by all the different objects in a class.

Final variable

- You can declare a variable in any scope to be final.
- The value of a final variable cannot change after it has been initialized. Such variables are similar to constants

Difference between instance & class variable



Instance Variable	Class Variable
Every object will have its own copy of instance variables, hence changes made to these variables through one object will not reflect in another object.	Class variables are common to all objects of a class, if any changes are made to these variables through object, it will reflect in other objects as well.
Instance variables are declared without static keyword.	Class variables are declared with keyword static
Instance variables can be used only via object reference.	Class variables can be used through either class name or object reference.



Java Operator



Operators



Unary

- ++, --



Arithmetic

- +, -, /, *, %



Relational

- <, >, ==, >=, <=, !=
- instanceof



Logical

- &&, ||, !



Bitwise

- &, |, ^, >>, <<, >>>

operator precedence



OPERATOR	TYPE	ASSOCIIVITY
() [] . ->		left-to-right
++ -- +- ! ~ (type) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right

OPERATOR	TYPE	ASSOCIIVITY
	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= = <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right



Demo on operators





Quiz





1

Which of the following is not a valid variable name?

- A. _blue B. 2blue C. blue\$ D. Blue

B. 2blue

2

Which of these class names best follows standard Java naming conventions?

- A. fooBar B. FooBar C. FOO_BAR D. F_o_o_B_a_r

B. FooBar

3

What is the value of tip after executing the following code snippet?

```
int meal = 5;  
int tip = 2;  
int total = meal + (meal > 6 ? ++tip : tip);
```

- A. 1 B. 2 C. 3 D. 6

A. 1



4

What is the output of the following code snippet?

```
int x = 10, y = 5;  
boolean w = true, z = false;  
x = w ? y++ : y--;  
w = !z;  
System.out.print((x+y)+" "+(w ? 5 : 10));
```

C. 11 5

- A. The code does not compile. B. 10 10 C. 11 5 D. 12 5

5

What will be the output of the program?

```
class Bitwise  
{  
    public static void main(String [] args)  
    {  
        int x = 11 & 9;  
        int y = x ^ 3;  
        System.out.println( y | 12 );  
    }  
}
```

C. 14

- A. 7 B. 0 C. 14 D. 8



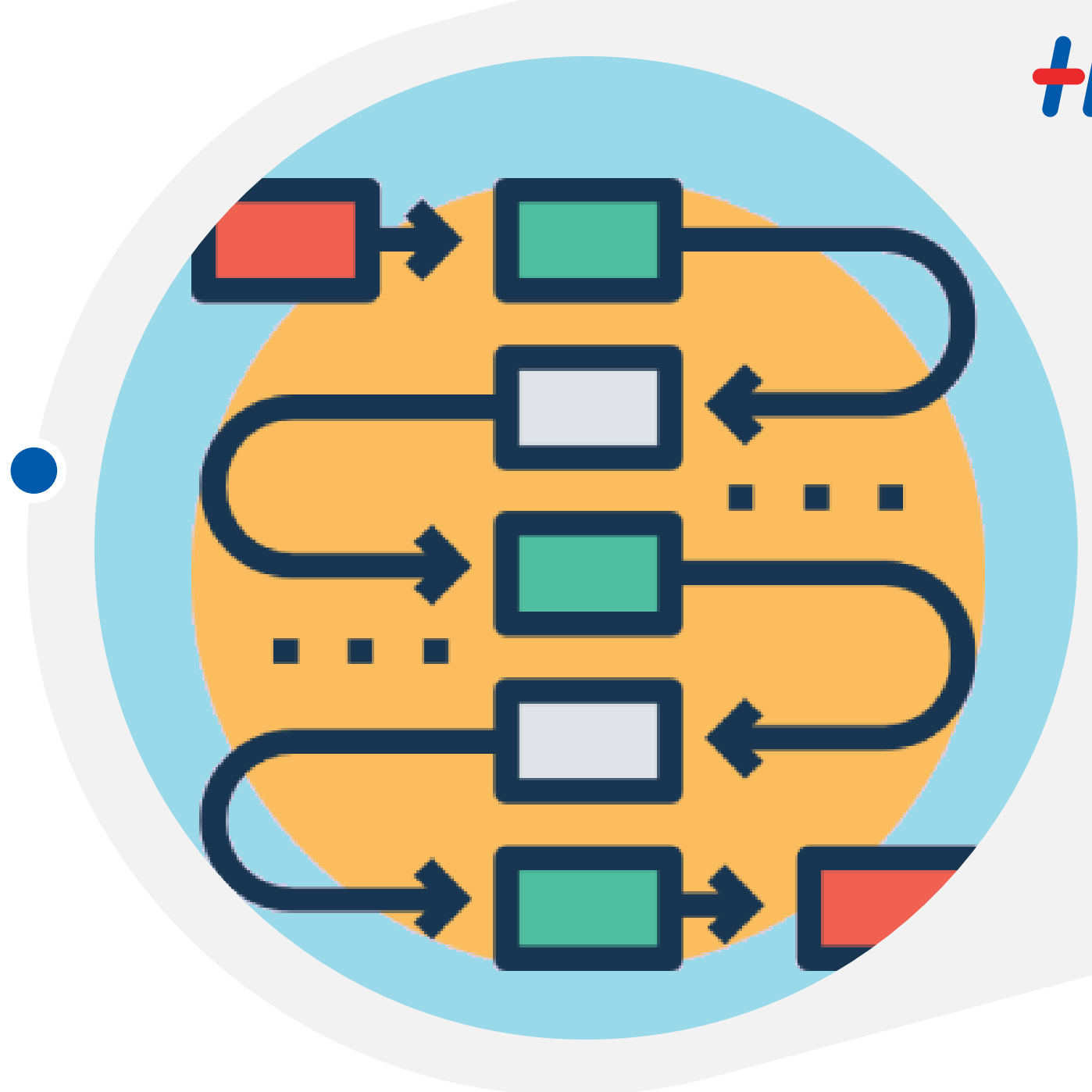
Control Statement



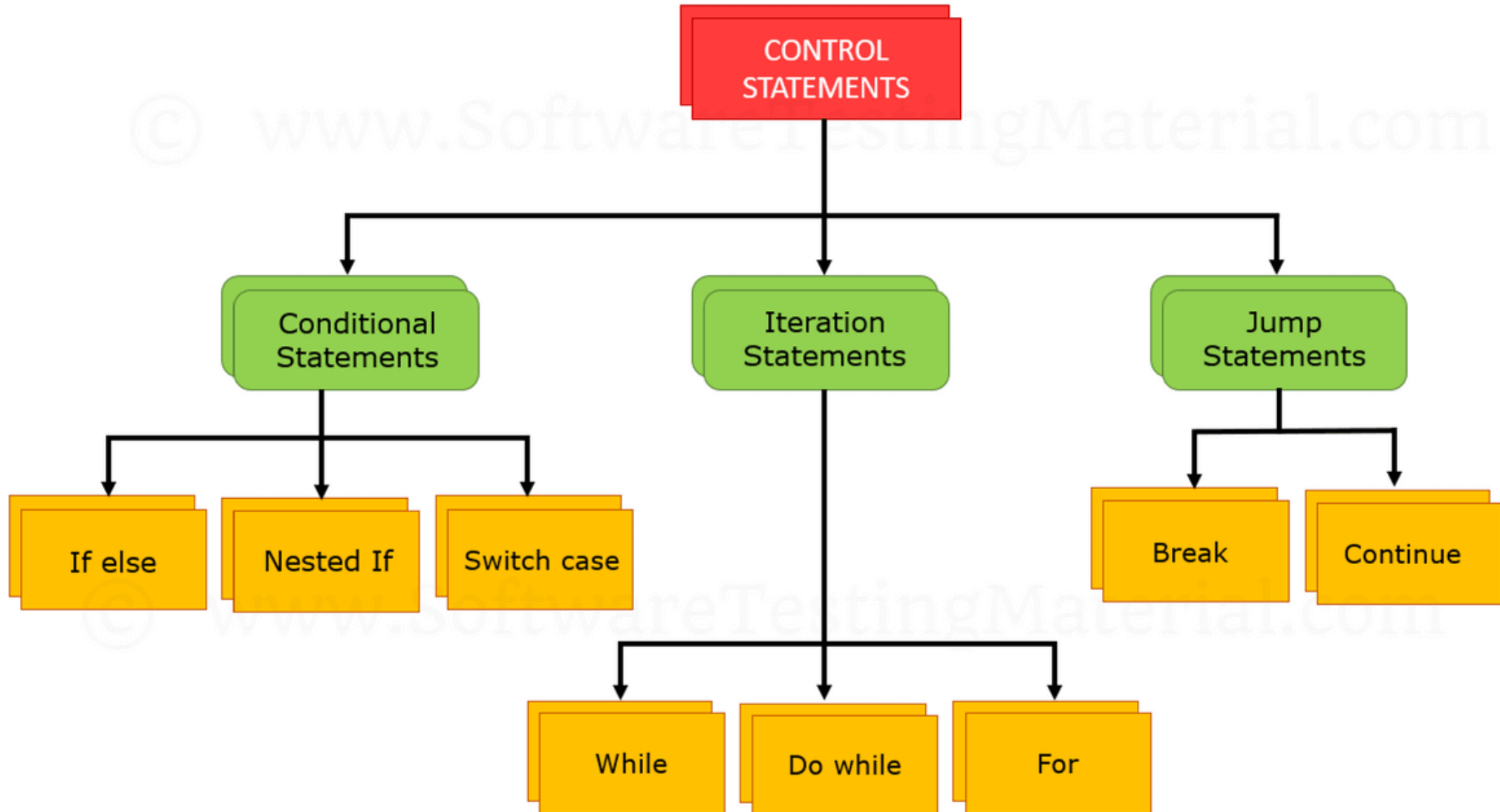
What is Control Statements?



- Control statements are the statements that change the flow of execution of statements.
- Boolean expressions are used as conditional expressions in a statement that alter the flow of control.
- Control Statements can be divided into three categories, namely
 1. Selection statements
 2. Iteration statements
 3. Jump statements



Control Statements



Decision-Making Statements



- Statements that determine which statement to execute and when are known as decision-making statements. The flow of the execution of the program is controlled by the control flow statement.

If Statement

```
if (condition)
{
    Statement 1;
    //executed if condition is true
}
Statement 2;
    //executed irrespective of the condition
```

Syntax

Output:
a is less than 10
Hello World!

```
public class Main
{
    public static void main (String args[])
    {
        int a = 15;
        if (a > 20)
            System.out.println ("a is greater than 10");
        else
            System.out.println ("a is less than 10");
        System.out.println ("Hello World!");
    }
}
```

Example

Nested if statement

- An if present inside an if block is known as a nested if block. It is similar to an if..else statement, except they are defined inside another if..else statement.

```
if (condition1)
{
    //Statement 1;
    if (condition2)
    {
        //Statement 2;
    }
    else
    {
        //Statement 3;
    }
}
```

Syntax

```
public class Main
{
    public static void main (String args[])
    {
        int s = 18;
        if (s > 10)
        {
            if (s % 2 == 0)
                System.out.println ("s is even and greater than 10!");
            else
                System.out.println ("s is odd and greater than 10!");
        }
        else{
            System.out.println ("s is less than 10");
        }
        System.out.println ("Hello World!");
    }
}
```

Example

Output:

```
s is even and greater than 10!
Hello World!
```




Demo





- A switch statement in java is used to execute a single statement from multiple conditions. The switch statement can be used with short, byte, int, long, enum types, etc.
- Must be noted while using the switch statement:
 - One or N number of case values can be specified for a switch expression.
 - Case values that cannot be duplicated. A compile-time error is generated by the compiler if unique values are not used.
 - The case value must be literal or constant. Variables are cannot be used.
 - Usage of break statement is made to terminate the statement sequence. It is optional to use this statement. If this statement is not specified, the next case is executed.

Switch statement



Example

```
public class Music{
    public static void main (String[]args) {
        int instrument = 4;
        String musicInstrument;
        // switch statement with int data type
        switch (instrument){
            case 1:
                musicInstrument = "Guitar";
                break;
            case 2: musicInstrument = "Piano";
                break;
            case 3: musicInstrument = "Drums";
                break;
            case 4: musicInstrument = "Flute";
                break;
            default: musicInstrument = "Invalid";
                break;
        }
        System.out.println (musicInstrument);
    }
}
```

Output:
Flute



Demo



Looping Statements



- Statements that execute a block of code repeatedly until a specified condition is met are known as looping statements. while, for, do...while, for...each are the three types of loop.

While

```
while (condition)
{
    statementOne;
}
```

Syntax

Output:

```
5
7
9
11
13
15
```

```
public class whileTest
{
    public static void main(String args[])
    {
        int i = 5;
        while (i <= 15)
        {
            System.out.println(i);
            i = i+2;
        }
    }
}
```

Example



Demo



Looping Statements



While

```
do{  
    //code to be executed  
}while(condition);
```

Syntax

Output:
20

```
public class Main  
{  
    public static void main(String args[])  
    {  
        int i = 20;  
        do  
        {  
            System.out.println(i);  
            i = i+1;  
        } while (i <= 20);  
    }  
}
```

Example



Demo



Looping Statements



for loop

```
for (initialization; condition; increment/decrement)
{
    statement;
}
```

Syntax

Output:

```
5
6
7
8
9
10
```

```
public class forLoop
{
    public static void main(String args[])
    {
        for (int i = 1; i <= 10; i++)
            System.out.println(i);
    }
}
```

Example



Demo





1

What does the following code output?

```
int singer = 0;  
while (singer)  
System.out.println(singer++);
```

- A. 0
- B. The code does not compile.
- C. The loops complete with no output.
- D. This is an infinite loop.

B. The code does not compile.

2

What is the result of the following code?

```
do {  
int count = 0;  
do {  
count++;  
} while (count < 2);  
break;  
} while (true);  
System.out.println(count);
```

- A. 2
- B. 3
- C. The code does not compile.
- D. This is an infinite loop

C. The code does not compile.

3

What is the output of the following program?

```
public class Test{
    public static void main(String []args){
        int i = 0;
        for(i = 0; i < 10; i++){break; }
        System.out.println(i);
    }
}
```

A. 1 B. 0 C. 10 D. 9

B. 0



4

what will be the output of the following program?

```
public class Test {
    public static void main(String[] args) {
        int i = 0;
        for (System.out.print("HI"); i < 1; i++)
            System.out.print(" Hexaware");
    }
}
```

A. HI Hexaware B. No Output C. Compile time error D. HELLO GEEKS

C. The code does not compile.

5

Which of the following iterates a different number of times than the others?

- A. `for (int k=0; k < 5; k++) {}`
- B. `for (int k=1; k <= 5; k++) {}`
- C. `int k=0; do { } while(k++ < 5)`
- D. `int k=0; while (k++ < 5) {}`



C. `int k=0; do { }
while(k++ < 5)`

6

What is the output of the following?

```
12: int result = 8;  
13: for: while (result > 7) {  
14: result++;  
15: do {  
16: result--;  
17: } while (result > 5);  
18: break for;  
19: }  
20: System.out.println(result);
```

- A. 5 B. 8 C. The code does not compile. D. exception at runtime.

C. The code does not compile.



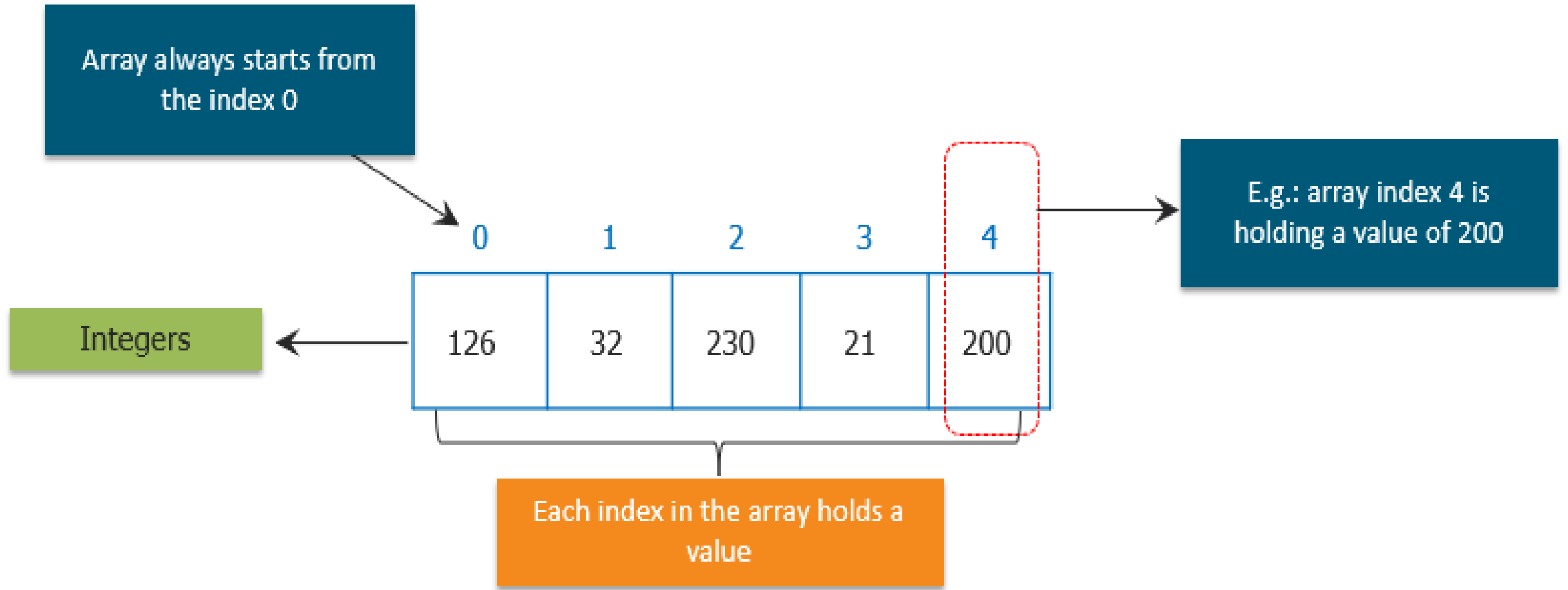
Array in java





- Java allows us to hold many objects of the same type using arrays.
- Array are utilized by the help of a loop to access the elements with their index.

Array Example

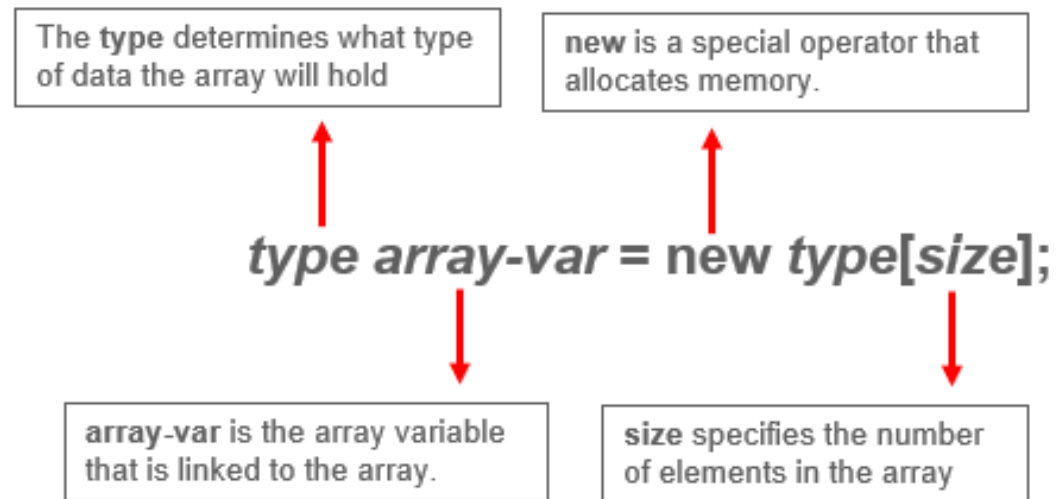


Fixed Size, Not Possible to extend the size



Array Declaration and creation

- Below three statements are valid Array declaration.
 1. `int[] a;`
 2. `int []x;`
 3. `int x[];`
- First declaration are prescribed, name and type are clearly separated
- while declaring array defining size is incorrect (`int[6] a;`)



Array creation



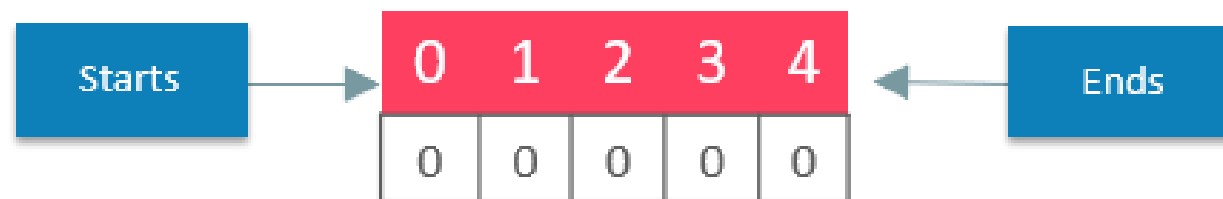
1

data type size of array

↓ ↓

```
int[] a= new int[5];
```

Index has to be given in
square brackets



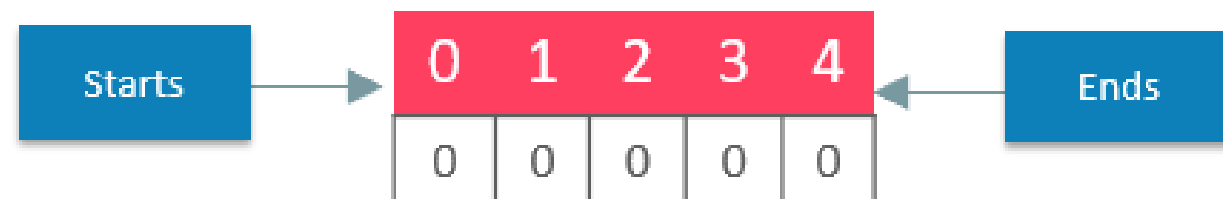
2

data type size of array

↓ ↓

```
int a[]= new int[5];
```

Index has to be given in
square brackets



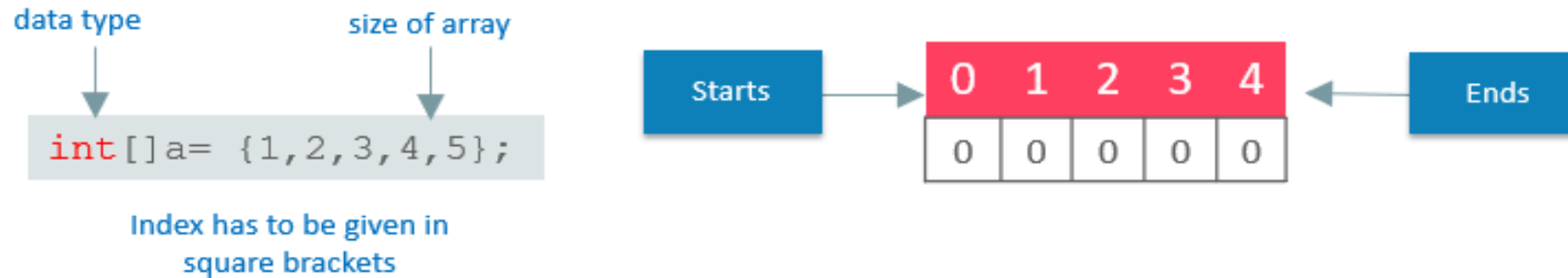
Array creation



The type determines what type of data the array will hold

type var-name[] = {value1, value2, value3, value4,...};

An array initializer is a list of comma-separated expressions surrounded by curly braces. The commas separate the values of the array elements



Example



Example

```
Class ArrayExample{
public static void main(String args[]) {
    int month_days[];
    month_days = new int[12];
    month_days[0] = 31;      month_days[1] = 28;
    month_days[2] = 31;      month_days[3] = 30;
    month_days[4] = 31;      month_days[5] = 30;
    month_days[6] = 31;      month_days[8] = 30;
    month_days[9] = 31;      month_days[10] = 30;
    month_days[11] = 31;
    System.out.println("April has " + month_days[3] + " days.");
}
}
```

- When we try to access `month_days[12] = 31;` we get

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 14
at ArrayExample.main(ArrayExample.java:14)

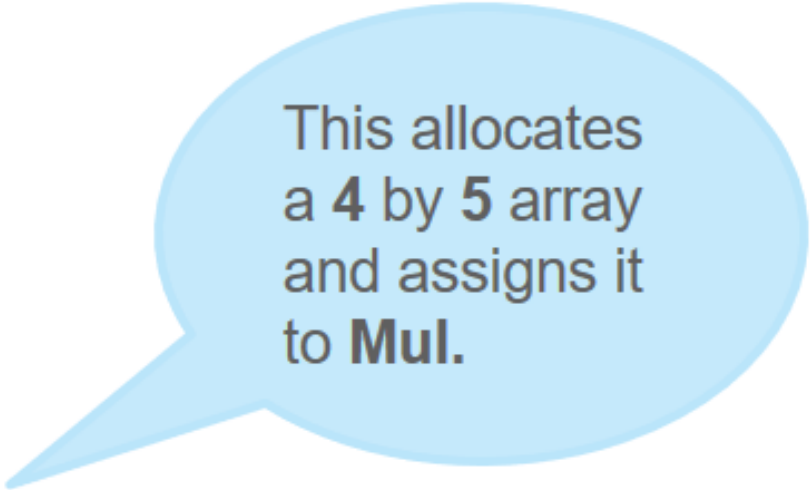


1D Array Demo



Declaring Multidimensional Array

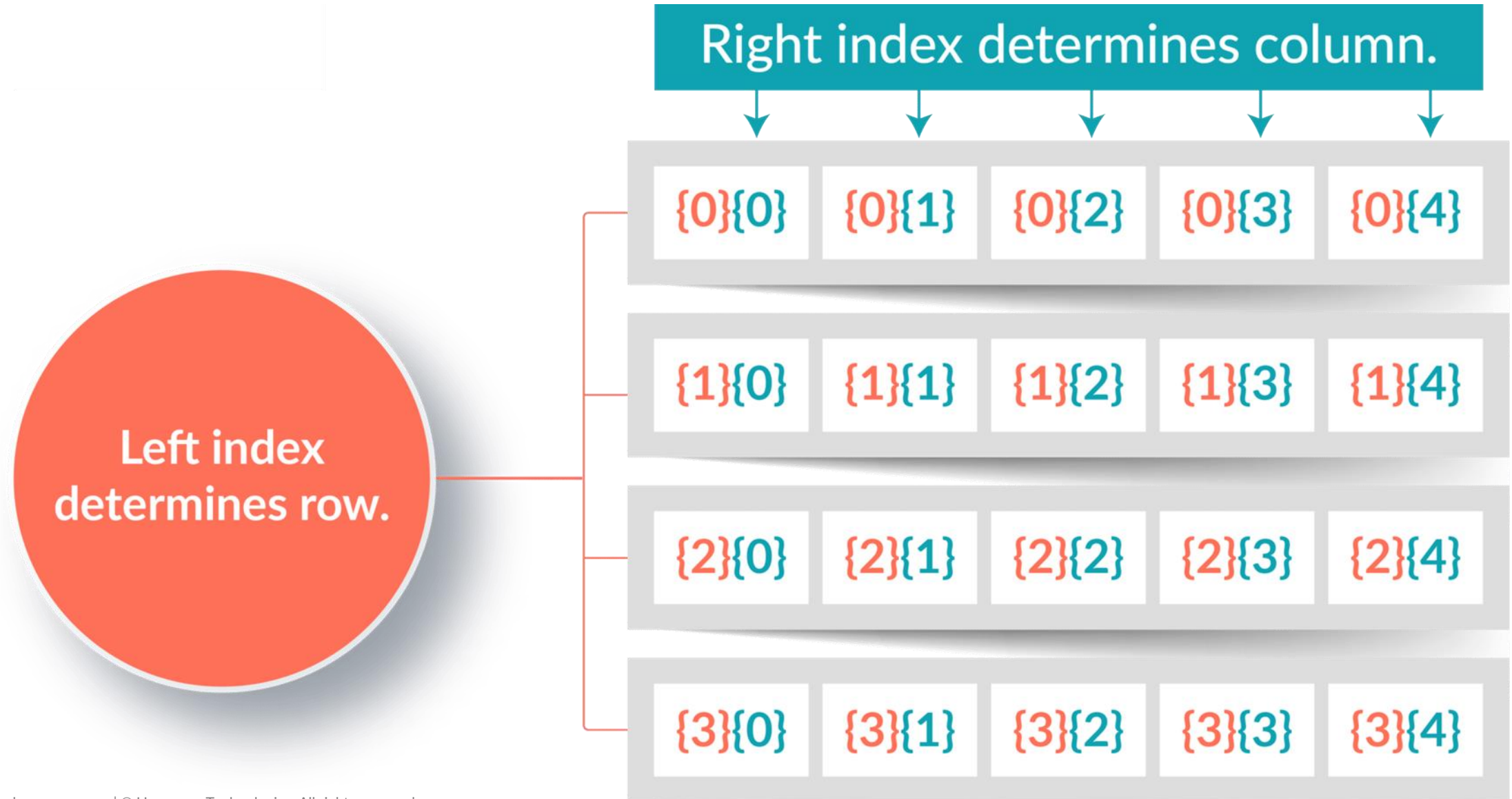
- To declare it, we must specify each additional index using another set of square brackets.



This allocates
a **4** by **5** array
and assigns it
to **Mul**.

```
int Mul[ ][ ] = new int[4][5];
```

2D Array Index



2D array Representation



```
int [][]a= new int [2][2];
```

	0	1
0	1	4
1	4	5

2 x 2 dimensional int array

```
char [][]a= new char[3][2];
```

	0	1
0	s	a
1	g	v
2	v	d

```
float [][]a= new float[5][5];
```

	0	1	2	3	4
0	2.2	3.4	5.0	3.3	1.2
1	7.8	9.0	1.1	2.9	5.5
2	2.0	3.0	7.8	9.8	9.9
3	5.7	6.6	8.8	5.3	2.7
4	1.8	4.4	7.6	1.0	1.1

5 x 5 dimensional float array

Example



Example

```
class Mul2D
{
    public static void main(String args[])
    {
        int mul2d[][]= new int[4][5];
        int i, j, k = 0;
        for(i=0; i<4; i++)
            for(j=0; j<5; j++) {
                Mul2D[i][j] = k;
                k++;
            }
        for(i=0; i<4; i++)
        {
            for(j=0; j<5; j++);
            System.out.print(mul2d[i][j] + " ");
            System.out.println();
        }
    }
}
```

output:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```



2D Array Demo



Jagged Array



- we can create a 2-D arrays with variable number of columns in each row.
- Each row in 2-D array itself an array.

```
int[][] matrix={  
    {1,2,3,4,5},  
    {5,6,7},  
    {6,7,8,9,3,1},  
    {4,5},  
    {5}  
}
```

1	2	3	4	5	
5	6	7			
6	7	8	9	3	1
4	5				
5					

```
matrix.length is 5  
matrix[0].length is 5  
matrix[1].length is 3  
matrix[2].length is 6  
matrix[3].length is 2  
matrix[4].length is 1
```

Enum



- An enum is a special "class" that represents a group of constants (unchangeable variables, like final variables).
- To create an enum, use the enum keyword (instead of class or interface), and separate the constants with a comma. Note that they should be in uppercase letters:

```
enum Level {  
    LOW,  
    MEDIUM,  
    HIGH  
}  
  
Level myVar = Level.MEDIUM;
```

Example

```
public class Main {  
    enum Level {  
        LOW,  
        MEDIUM,  
        HIGH  
    }  
  
    public static void main(String[] args) {  
        Level myVar = Level.MEDIUM;  
        System.out.println(myVar);  
    }  
}
```




Demo



Jagged Array



Example

```
class JaggedDemo {  
    public static void main(String[] args) {  
        int r = 5;  
        // Declaring 2-D array with 5 rows  
        int arr[][] = new int[r][];  
        // Creating a 2D array such that first row has 1 element,  
        // second row has two elements and so on.  
        for (int i=0; i<arr.length; i++)  
            arr[i] = new int[i+1];    // Initializing array  
        int count = 0;  
        for (int i=0; i<arr.length; i++)  
            for(int j=0; j<arr[i].length; j++)  
                arr[i][j] = count++;  
        // Displaying the values of 2D Jagged array  
        System.out.println("Contents of 2D Jagged Array");  
        for (int i=0; i<arr.length; i++)  
        {  
            for (int j=0; j<arr[i].length; j++)  
                System.out.print(arr[i][j] + " ");  
            System.out.println();  
        }  
    }  
}
```

Output

```
0  
1 2  
3 4 5  
6 7 8 9  
10 11 12 13 14
```



1

Print the second item in the cars array.

```
String[] cars = {"Volvo", "BMW", "Ford"};  
System.out.println(_____);
```

- A. cars[1] B. cars[2] C. cars[0] D. cars[3]

A. cars[1]

2

What is the result of running the following as java Copier?

```
package duplicate;  
public class Copier {  
    public static void main(String... original) {  
        String... copy = original;  
        System.out.println(copy.length + " " + copy[0]);  
    }  
}
```

- A. 0 B. 0 followed by an exception C. 1 followed by an exception
D. The code does not compile.

D. The code does not compile.



What is the result of running the following program?

```
1: package fun;  
2: public class Sudoku {  
3: static int[][] game = new int[6][6];  
4:  
5: public static void main(String[] args) {  
6: game[3][3] = 6;  
7: Object[] obj = game;  
8: obj[3] = "X";  
9: System.out.println(game[3][3]);  
10: }  
11: }
```

- A. X
- B. The code does not compile.
- C. The code compiles but throws a `NullPointerException` at runtime.
- D. The code compiles but throws a different exception at runtime.

D. The code compiles but throws a different exception at runtime.

What will be the output of the following Java code?

```
class array_output
{
    public static void main(String args[])
    {
        int array_variable[][] = {{ 1, 2, 3}, { 4 , 5, 6}, { 7, 8, 9}};
        int sum = 0;
        for (int i = 0; i < 3; ++i)
            for (int j = 0; j < 3 ; ++j)
                sum = sum + array_variable[i][j];
        System.out.print(sum / 5);
    }
}
```

b) 9

a) 8 b) 9 c) 10 d) 11

What will be the output of the following Java code?

```
class array_output
{
    public static void main(String args[])
    {
        char array_variable [] = new char[10];
        for (int i = 0; i < 10; ++i)
        {
            array_variable[i] = 'i';
            System.out.print(array_variable[i] + "");
        }
    }
}
```

- a) 1 2 3 4 5 6 7 8 9 10
- b) 0 1 2 3 4 5 6 7 8 9 10
- c) i j k l m n o p q r
- d) i i i i i i i i i i

b) 9



References



1. <https://www.geeksforgeeks.org/java/>
2. <https://www.tutorialspoint.com/java/index.htm>
3. <https://www.edureka.co/blog/java-tutorial/>
4. <https://www.javatpoint.com/java-tutorial>
5. <https://www.programiz.com/java-programming>



Thank you

Innovative Services



Passionate Employees



Delighted Customers

