

# Core Java : Fundamentals

Getting started with Java

Team Emertxe





Table of contents

# Table of contents

- What is Java
- History of Java
- Features of Java
- Versions of Java
- Installing software
- JRE & JDK
- Running a Hello World Program
- Variable and Data types
- Java Keywords

# Table of contents

- Naming convention
- Java comments
- More About HelloWorld program
- Enum types
- Operator
- The control flow
- Array
- JVM Architecture



What is Java?

# Defining Java

- Java is an object oriented programming language.
- Developed by Sun Microsystems (Currently owned by Oracle Corporation Inc.)
- First Version Release: 1996 JDK 1.0(The current version is JDK1.8)
- People Associated with Java:
  - James Gosling
  - Patric Naughton
  - Mike Sheridan



# Java History

# Java History



- James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991.
- Java was originally designed for embedded devices.
- The language was initially called Oak.
- Later the project went by the name Green and was finally renamed Java.




# Features of Java



# Features of Java



Different versions of Java



# Different Java's versions



- JDK Alpha and Beta (1995)
- JDK 1.0 (January 23, 1996)
- JDK 1.1 (February 19, 1997)
- JDK 1.2 (December 8, 1998)
- JDK 1.3 (May 8, 2000)
- JDK 1.4 (February 6, 2002)
- JDK 1.5 (September 30, 2004)
- JDK 1.6 (December 11, 2006)
- JDK 1.7 (July 28, 2011)
- JDK 1.8 (March 18, 2014)

# Difference between Java Versions



- JDK Alpha and Beta: Alpha and Beta Java public releases had highly unstable APIs. The supplied Java web browser was named WebRunner.
- JDK 1.0: Originally called Oak. The first stable version, JDK 1.0.2, is called Java 1.
- JDK 1.1: AWT, inner class, JDBC, RMI, Java Beans, reflection and JIT compiler
- JDK 1.2: Codename Playground. Strictfp keyword, Swing, Collection framework

# Difference between Java Versions



- JDK 1.3: JNDI
- JDK 1.4: assert keyword, nio(non blocking io) package.
- JDK 1.5: Code name tiger. Generics, Annotations, Autoboxing, Enumerations, Varargs, for-each loop,static imports.
- JDK 1.6: Code name Mustang. Support for older win9X dropped. JDBC 4.0, support for pluggable annotations. JVM improvements: compiler performance optimizations, upgrades to existing garbage collection algorithms.

# Difference between Java Versions



- JDK 1.7: Code name dolphin.
  - JVM supports for dynamic languages.
  - Strings in switch block
  - The new package are `java.nio.file` and `java.nio.file.attribute`
- JDK 1.8:
  - Classes and interfaces have been added to the `java.util.concurrent` package.
  - The class `java.net.URLPermission` has been added.
  - The JDBC-ODBC Bridge has been removed.
  - Parallel Array Sorting
  - New `java.util.stream` package

# Installing Java Software





# Software Download



The screenshot shows the Oracle Java SE Downloads page. The browser address bar displays [www.oracle.com/technetwork/java/javase/downloads/index.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html). The Oracle logo is at the top left, with navigation links for Products and Services, Solutions, Downloads, Store, Support, Training, Partners, and About. A breadcrumb trail shows the path: Oracle Technology Network > Java > Java SE > Downloads. The main content area is titled "Java SE Downloads" and features four tabs: Overview, Downloads (selected), Documentation, Community, Technologies, and Training. Below the tabs are four categories: Latest Release, Next Release (Early Access), Embedded Use, and Previous Releases. The "Latest Release" category is highlighted with a red box and contains a "Java Platform (JDK) 7u13" download button. Other categories include "JavaFX 2.2.5" and "JDK 7 + NetBeans". A left sidebar lists various Java-related topics.

www.oracle.com/technetwork/java/javase/downloads/index.html

g Started 1 Requests + eHow Spark Latest Headlines NoFollow Link Checker... su.pr - Drive More Tra... BizSugar Show Non f

**ORACLE** Sign In/Register for Account Help Select Country/Region Communities I am a...

PRODUCTS AND SERVICES SOLUTIONS DOWNLOADS STORE SUPPORT TRAINING PARTNERS ABOUT

Oracle Technology Network > Java > Java SE > Downloads

Overview **Downloads** Documentation Community Technologies Training

**Java SE Downloads**

Latest Release Next Release (Early Access) Embedded Use Previous Releases

 **DOWNLOAD**

**Java Platform (JDK) 7u13**

 **DOWNLOAD**

**JavaFX 2.2.5**

 **DOWNLOAD**

**JDK 7 + NetBeans**

Java SE  
Java EE  
Java ME  
Java SE Support  
Java SE Advanced & Suite  
Java Embedded  
JavaFX  
Java DB  
Web Tier  
Java Card  
Java TV  
New to Java  
Community  
Java Magazine



# Environment Variables



- System variables
  - Specific to the system
- User defined variables
  - Specific to only user

# Path

- PATH is used to define where the system can find the executables (.exe) files
- Whenever you type something into a command prompt, if it is not a system command, it will search in that directory that you have already added in the PATH variable.

# Classpath

- CLASSPATH is used to specify the location of .class files.
- CLASSPATH is used by the JVM that tells the JVM where to find the class libraries, including user-defined classes and libraries to import or interpret at compile time or run time.
- The CLASSPATH is a list of locations, where the JVM will search for classes (jars and directories).

JDK and JRE



## Java Development Kit

Development Tools

Compiler

Documentation

## Java Runtime Environment

Java Virtual Machine

Libraries

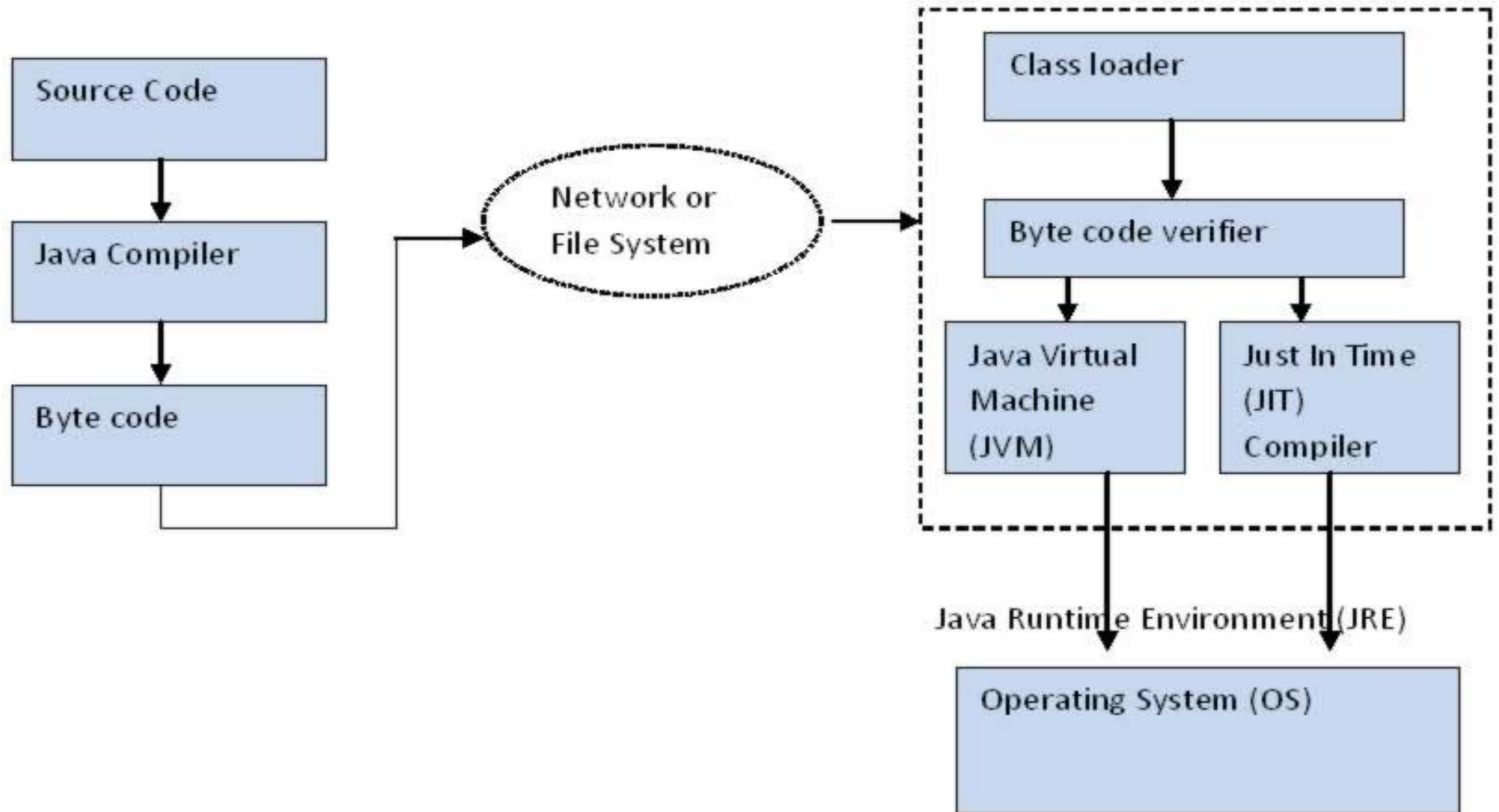


## Difference between JDK & JRE



- JDK provides tool for developer to develop Java application.
- JRE provides tools to test the Java application already developed by using JDK tools.

# JRE





# A “Hello World” Java Program



# Hello World Java Program

```
Class HelloWorld{  
    public static void main(String[] args){  
        System.out.println("");  
    }  
}
```

- Compilation: `javac HelloWorld.java`
- Execution: `java HelloWorld`



# Compilation in Java

- Compiling a Java program means taking the programmer-readable text in your program file (also called source code) and converting it to byte codes, which are platform-independent instructions for the Java VM.

- `Javac HelloWorld.java`



# Execution of a Java Program



- Once you have successfully compiled your Java source code, you can invoke the Java VM to run your application's byte-code:

- Java Helloworld



# Variable & Data Types



# Variable & Data Types

- Variable: It stores the data
- Types of variable:
  - Static variable: Declared in the class with the static keyword
  - Instance variable: Declared in the class without static keyword
  - Local Variable: Declared inside the method

# Data Types



- Data Types: It defines the size and behavior of data.
- There are two types of data type in Java:
  - Primitives Type  
e.g-byte,short,int,long,char,float,double,boolean
  - Class Type/Object Type  
e.g-String, array and all user defined

# Static vs Instance Variable(Non-static)



- Static is for the cases where you don't want to have copy for each instance.
- A static variable is one per Class, every object of that class shares the same Static variable.
- A static variable is initialized when the JVM loads the class
- Instance variables are for the cases where you want separate copy for each instance of object.
- An instance variable is one per Object, every object has its own copy of instance variable.



# Primitives



# Primitives

- byte : 0
- short : 0
- int : 0
- long: 0L
- char:\0000u
- float: 0.0f
- double: 0.0D
- boolean: false

# Class Type Data Type

- Java.lang.String[We will discuss it later in different chapter]
- Array[coming next]
- All user defined classes created by programmer.

Keywords



# Keywords

- There are 48 keywords in Java:
  - Byte, short, int, char, long, float, double, boolean, if, else, for, etc.
- 2 Reserved words: const, goto
- 3 Literals: true, false, null
- Keywords can't be used as identifiers(class name, variable name, package name, etc).

# Naming Convention



# Java Naming Convention



- Java naming convention allow a-z and A-Z letters.
- Two special character: \$ and \_
- Ten digits: 0-9

# Java Naming Convention



- The class will follow ProperCase
- The variable name will follow camelCase.
- Method name will follow camelCase.
- Package name will follow the company url names such as com.emertxe.
- Constant naming convention will follow CAPITAL\_CASE



# Java Comments

- Single line comments:

```
//-----
```

- Multiple line comments:

```
/*
```

```
this is multiple line comments
```

```
you are at Emertxe information technologies
```

```
*/
```

# Java Comments

- Java doc comment:

```
/**
```

```
*this is java documentation comment
```

```
*you are at Emertxe information technologies
```

```
*/
```

# More About HelloWorld Java Program



- ```
class Test{  
    public static void main(String[] args){  
  
        System.out.println("Hello World");  
    }  
}
```

# Java Source Files

- Java source files can be saved with any names.
- If the class is declared public then you have to save the file with the same name.
- If the class is not declared as public then you are free to save the file with any name.

# Java Source Files

- We compile the source file but run the byte code.
- You can compile the source file with any name but you have to run the byte code which name is same as the actual class name created by the programmer.

# Compiling & Running



- Compiling: `javac Test.java`
- Running: `java Test`

# Enum Type

- An enum type is a special data type that enables for a variable to be a set of predefined constants.
- The variable must be equal to one of the values that have been predefined for it.
- Common examples include compass directions (values of NORTH, SOUTH, EAST, and WEST) and the days of the week.

# Enum Type

- The names of an enum type's fields are in uppercase letters.
- ```
public enum Day {  
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,  
    THURSDAY, FRIDAY, SATURDAY  
}
```
- You should use enum types any time you need to represent a fixed set of constants.



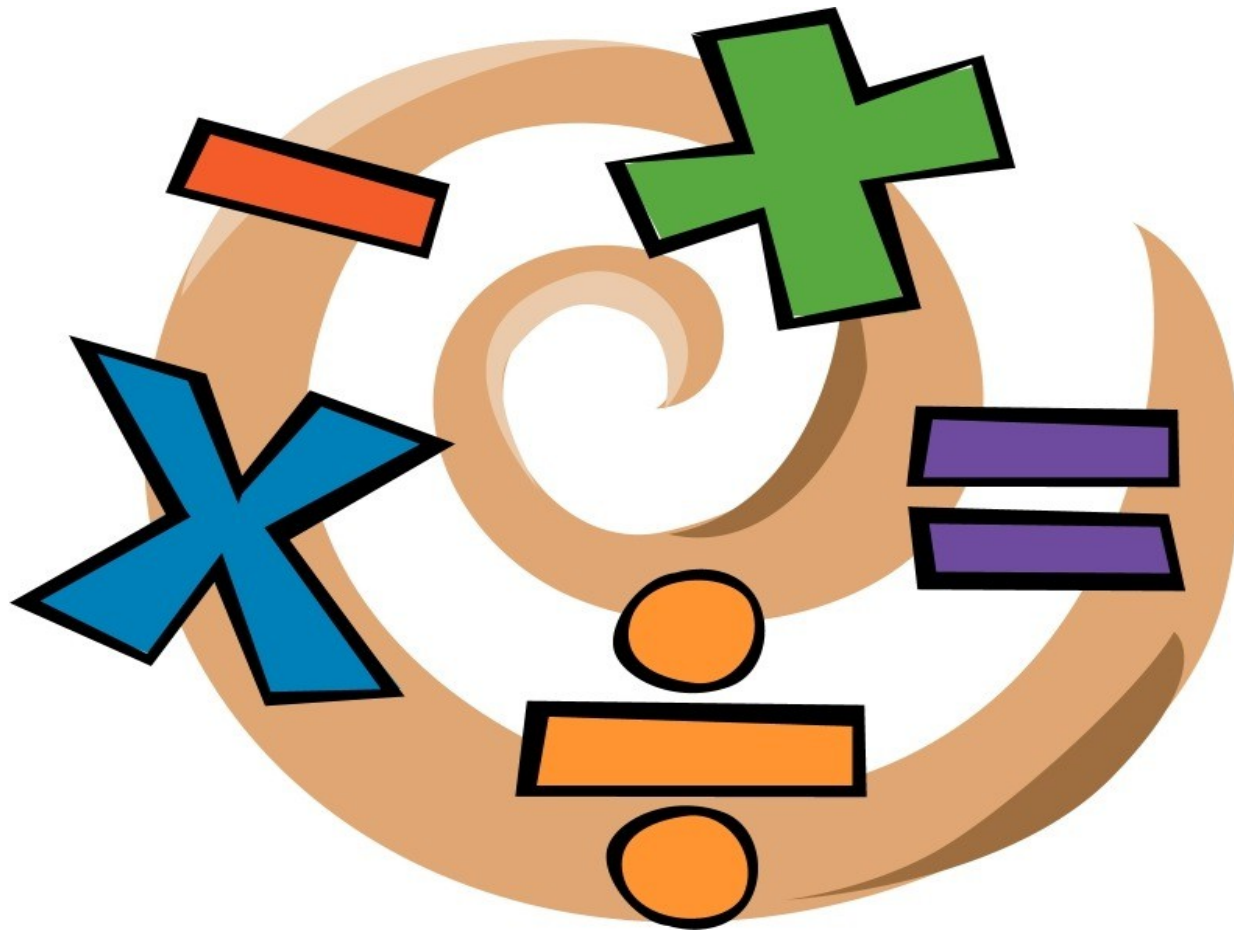


# Enum Type



- Write a program to implement enum for the weekday.

# Operators



# Operators

- Arithmetic operator
- Relational operator
- Assignment operator
- Logical operator
- Bit-wise operator
- Right shift operators  $\gg$  and  $\ggg$
- Left Shift operator  $\ll$

# Arithmetic Operators

- The unary arithmetic operators.(+, -)
- The increment and decrement operator(++ , --)
- Basic arithmetic operators(\* and /)
- The modulo operators(%)
  - The modulo operator gives the value that is the remainder of a division.
  - `int x = 11%3;`

# Relational Operators

- Comparison operators
  - $>, <, >=, <=, !=$

# Assignment Operator



- Assignment operator (=)
- Shot-circuit assignment operator:
  - $x = x + y$
  - $x += y$

# Logical Operators

- && AND
- || OR
- The boolean inversion operator(NOT): !



# Bit-wise operator



- The AND operator:  $\&$
- The OR operator:  $|$
- The XOR operator:  $\wedge$
- Complement operator  $\sim$



# Bit-wise Operator

A	B	A   B	A & B	A ^ B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

# Advance Operators

- Shortcut if-else Operator: ?:
  - `a = x ? b : c;`
- The instanceof Operator: The instanceof operator determines if a given object is of the type of a specific class.

# Questions



- Write Java programs to implement all the operators.

# Operator Precedence in Java



- Java has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators. For example, multiplication and division have a higher precedence than addition and subtraction. Precedence rules can be overridden by explicit parentheses.

# Precedence Order

When two operators share an operand the operator with the higher precedence goes first. For example,  $1 + 2 * 3$  is treated as  $1 + (2 * 3)$ , whereas  $1 * 2 + 3$  is treated as  $(1 * 2) + 3$  since multiplication has a higher precedence than addition.

# Associativity

- When an expression has two operators with the same precedence, the expression is evaluated according to its associativity.
- For example  $x = y = z = 17$  is treated as  $x = (y = (z = 17))$ , leaving all three variables with the value 17, since the  $=$  operator has right-to-left associativity (and an assignment statement evaluates to the value on the right hand side).
- On the other hand,  $72 / 2 / 3$  is treated as  $(72 / 2) / 3$  since the  $/$  operator has left-to-right associativity.

# Operator Precedence in Detail



- Postfix: `expr++ expr--`
- Unary: `++expr --expr +expr -expr ~ !`
- Multiplicative: `* / %`
- Additive: `+ -`
- Shift: `<< >> >>>`
- Relational: `< > <= >= instanceof`
- Equality: `== !=`

# Operator Precedence in Detail



- Bitwise AND:  $\&$
- Bitwise exclusive OR:  $\wedge$
- Bitwise inclusive OR:  $|$
- Logical AND:  $\&\&$
- Logical OR:  $||$
- Ternary:  $? :$
- Assignment:  $= \ += \ -= \ *= \ /= \ \% = \ \&= \ \wedge = \ |= \ \<\< = \ \>\> = \ \>\>\> =$



# Flow Control in Java



# Flow Control

- Selection statement
  - If statement
  - If-else statement
  - If -else if statement
  - If-else if-else
  - Switch statement

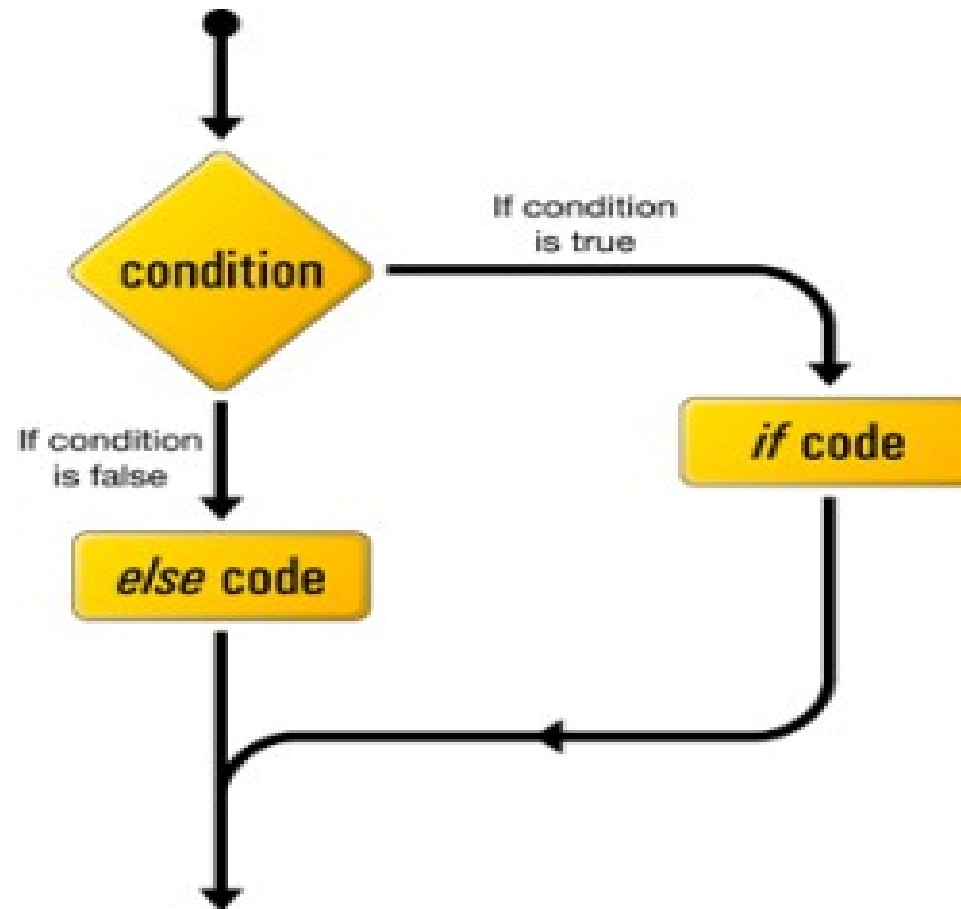
# Flow Control

- Iteration Statement
  - While loop
  - Do-while loop
  - For loop
  - For-each loop

# Flow Control

- Block breaker statement
  - Continue statement
  - Break statement

# The if Construct



# The if Construct

- The if construct allows the execution of a single statement.
- `if( <expression> ) {`  
    `// if <expression> returns true, the statements in this`  
    `// blocks are executed.`  
    `}`
- If there is only one statement in the block, the curly braces are not mandatory, but it is a good programming practice to use the curly braces regardless of whether there is one or more statements to execute.

# The if Construct

- Now consider the case in which we don't use the curly braces:
  1. `if ( x > 0 )`
  2. `System.out.println("x is greater than zero.");`
  3. `System.out.println("Who cares what x is.");`
- Line 2 will be executed only if x is greater than 0, whereas line 3 will always be executed independent of the value of x.

# The if Construct

- `if(x=y)` is illegal whereas the legal form is `if(x==y)`.
- However, if `x` and `y` are boolean, then `if(x=y)` will compile:

- `boolean b1 = false;`

`boolean b2 = true;`

`if(b1=b2){`

`System.out.println("The value of b1: " + b1);`

`}`

- The value of `b1`: `true`



# The if-else Construct



- If a condition is true, the first block of code will be executed, otherwise the second block of code will be executed.

- `if ( x > 0 ) {`

```
System.out.println("x is greater than zero.");
```

```
}
```

```
else {
```

```
System.out.println("x is not greater than zero.");
```

```
}
```

# The if-else if Construct



- With the if-else if construct you can handle multiple blocks of code, and only one of those blocks will be executed at most.
- Note that in an if-else if construct, the expressions will be tested one by one starting from the top. If an expression returns true, the block following the expression will be executed and all the following else if blocks will be skipped.
- Also note that it is possible that no block will be executed, a possibility that does not exist with the if-else construct.
- The syntax for the if-else construct:

# The if-else if Construct



- `if( <expression1> ) {`  
    `// if <expression1> returns true, statements in this block are executed.`  
    `}`  
    `else if ( <expression2> ) {`  
        `// if <expression1> is false and <expression2> is true,`  
        `then statements in this block will be executed.`  
    `}`  
    `else if (<expression3>) {`  
        `// if <expression1> is false, and <expression2> is false, and <expression3> is`  
        `true, then statements in this block`  
        `will be executed.`  
    `}`

# The if-else if-else Construct



- Java does offer a construct that enables you to handle multiple blocks of code and ensure that one of them will certainly be executed.

# The if-else if-else Construct



- `if( <expression1> ) {`  
    `// if <expression1> returns true, statements in this block are executed.`  
    `}`  
    `else if (<expression2>) {`  
        `// if the top level expressions are false then this will be executed.`  
        `}`  
    `else if (<expression3>) {`  
        `// if the all top level expressions are false then this will be executed.`  
        `}`  
    `else {`  
        `// if the all top level expressions are false then this will be executed.`  
        `}`

# Summary: If Statement



- Keep in mind that the condition in the if or else if statement can be a compound condition, However, the condition should always evaluate to a boolean. Such as:

```
if(i > 2 && j < 100)
```

# The switch Statement



- The switch statement is used to make the choices for multiple blocks with the possibility of executing more than one of them.
- `switch (x){`
  - `case 5:`  
`System.out.println("The value of x is 5." );`  
`break;`
  - `case 4:`  
`System.out.println("The value of x is 4." );`
  - `default:`  
`System.out.println("The value of x is default.");`

# The Switch Statement



- The default label could go anywhere in the switch block.
- If there is no break after the statement inside the switch block then switch will have fall through while execution.
- However, a simple mathematical expression inside the parentheses is fine, such as the following:

`switch ( x+ y ) or switch(x++)`

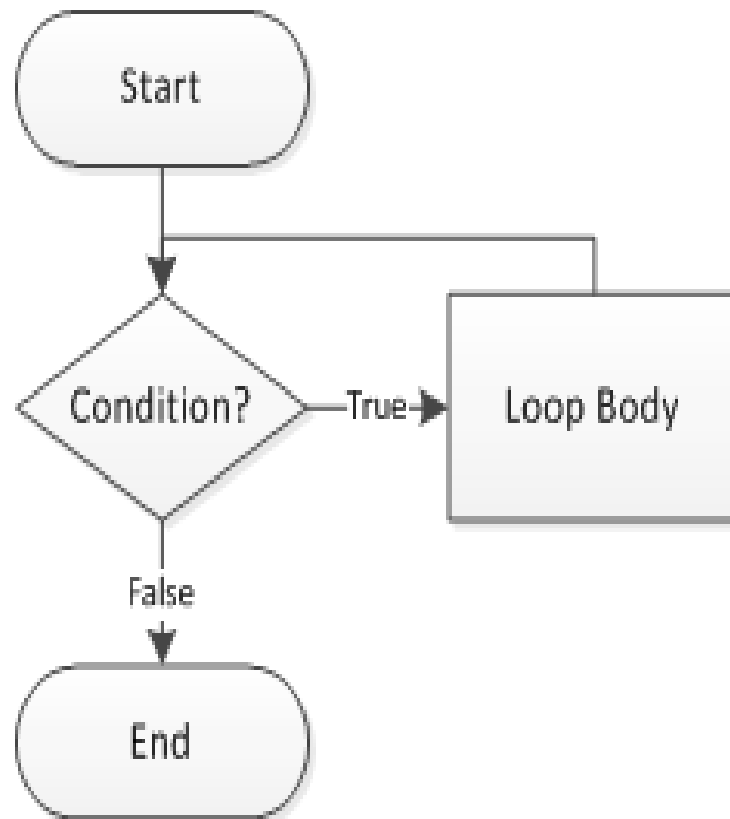


# The Switch Statement



- So, the legal argument type of a switch statement is int, or any other type that can be promoted to int: byte, short, or char.
- The legal argument types of a switch statement are byte , short char , int , enum and String
- ```
byte x=5;  
switch(x){  
case 5 : System.out.println("five");  
case 130 : System.out.println("one thirty"); //compiler error  
}
```
- The above code will result into compile time error because the compiler will look at 130 as an int, which can't fit into a byte.

# While Loop



# While Loop

- The while loop construct handles the situation in which a block is executed for the first time only when a condition is true.
- After execution the condition is checked again, and as long as the condition stays true, the block is executed repeatedly.
- `while ( <expression> ) {`  
    // if the <expression> is true, execute the statements  
    // After the execution, go back to check the condition again.  
}

# While Loop

- When the expression in the while parenthesis returns false, the execution control falls to the first statement immediately after the while block.

# Do-While Loop

- The do-while loop construct says: go ahead and execute the block for the first time, and then start checking the condition.
- ```
do {  
    // Execute the statements in this block.  
} while ( <expression> );
```
- When the execution control arrives at do, it enters the do block.
- After executing the statements in the do block, the expression specified by <expression> is executed.

# Do-While Loop



- If it returns true, the control goes back to the do statement and the do block is executed again.
- When the expression returns false, the control goes to the first statement immediately after the while statement.

# The for Loop

- The for loop is the most sophisticated of all the loop constructs and provides a richer functionality as a result.
- `for ( <statement>; <test>; <expression> ) {`  
    // if the <test> is true, execute the block.  
    }
- `int i;`  
    `for ( i=0; i < 3; i++) {`  
        `System.out.println("The value of i: " + i );`  
    }

# The For Loop

- `for (i; i<5;i++)// compile time error`
- `for (i=i+1; i<5;i++)// fine`
- `for( i++,j++ ;i+j < 5 ; i++)// fine`
- `for ( int i = 0, int j = 0; i+j < 5 ; i+  
+);//incorrect`
- `for (int i = 0; i = i + 1; i + j < 5;i++ )//incorrect`



# The For Loop

- Any of the three components in the parentheses of for() may be omitted, but the semicolons cannot be.
- ```
for(; ;) {  
    System.out.println ("Round and round we go for ever.");  
}
```

# The for-each Loop



- The for-each loop construct, introduced in J2SE 5.0.
- Easier and less error prone to iterate through the elements of an array and any other collection.
- ```
for (<variable> : <collection>) {  
    // the block code  
}
```

# The for-each Loop

- ```
int[] myArray = new int[3];  
myArray[0]= 10;  
myArray[1] = 20;  
for(int i : myArray) {  
    System.out.println (i);  
}
```

# The continue Statement



- If the continue statement is in the while, or do-while, block, control jumps to the boolean condition in the parentheses of while.
- If the continue statement is in the for block, control jumps to the <expression> in the for(<statement>; <test>; <expression>) statement.
- Note that the continue statement can be used only inside a loop: while, do-while, for, or for-each.

# The continue Statement



- ```
for ( int i = 0; i < 5; i++ ) {  
    if ( i == 3 ) continue;  
    System.println ( "The value of i is " + i );  
}
```
- The continue statement is only valid inside a loop. If you write it outside the loop, your program will generate a compiler error.

# The continue Statement



- In case of nested loops, you might need to specify from which loop you need to continue the next iteration. This is accomplished by using the labeled continue statement.
- OuterLoop: for ( int i = 3; i >0; i-- ) {  
    for (int j = 0; j<4; j = j + 1) {  
        System.out.println ( "i=" + i + " and j=" + j);  
        if ( i == j ) continue OuterLoop;  
    }  
}

# The break Statement



- The continue statement can only be used in a loop block, whereas a break statement can be used in a loop or in a switch statement.
- The break statement may also be used in the nested blocks.

# The break Statement



- ```
for ( int i = 3; i >0; i-- ) {  
    for (int j = 0; j<4; j++) {  
        System.out.println ( "i=" + i + " and j=" + j);  
        if ( i == j ) break;  
    }  
}
```
- In case of nested loops, you might need to tell from which loop you want to break.



# The break Statement



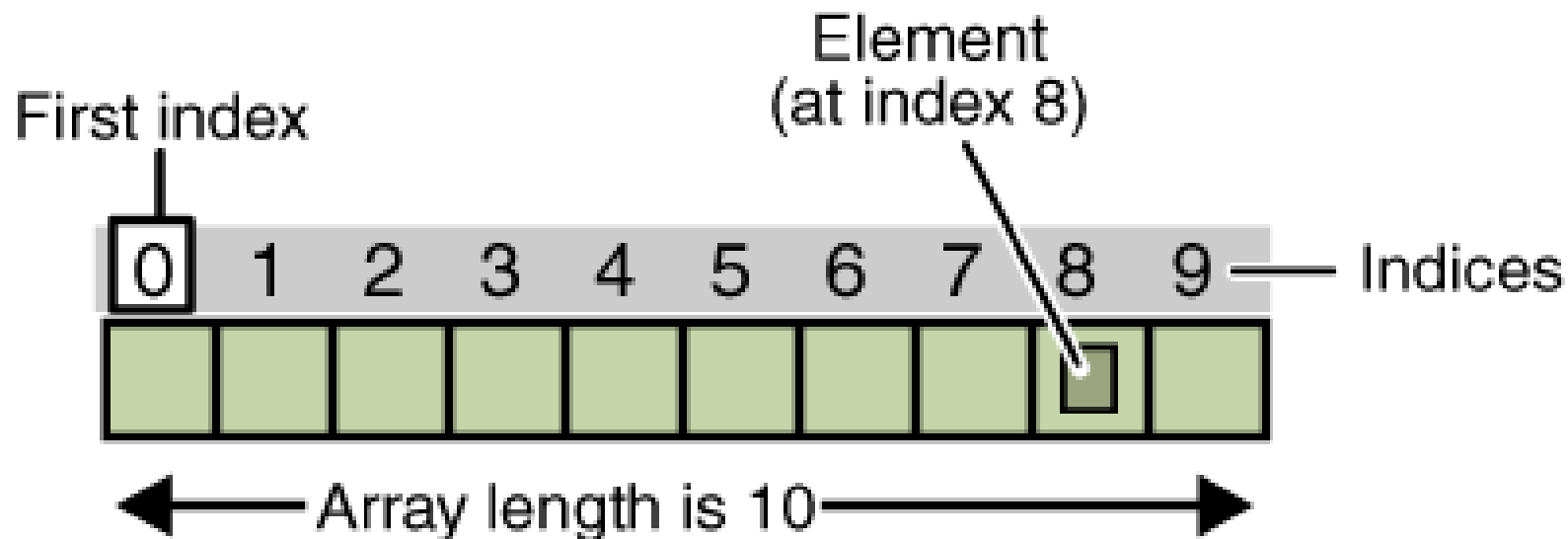
- OuterLoop: for ( int i = 3; i >0; i-- ) {  
    for (int j = 0; j<4; j = j + 1) {  
        System.out.println ( "i=" + i + " and j=" + j);  
        if ( i == j ) break OuterLoop;  
    }  
}

# Array

- Array in Java is an object that is used to store multiple variables of same type. These variables may be primitives or non-primitives.
- An array is an object; it is created with new.
- All array subscripts begin at 0.
- Declaring an array:
  - `Int [] arr = new int[5];`
  - `String[] arr;`
  - `arr = new String[5];`

# Array

- Initialize an array element.
- Create an array with initial values.



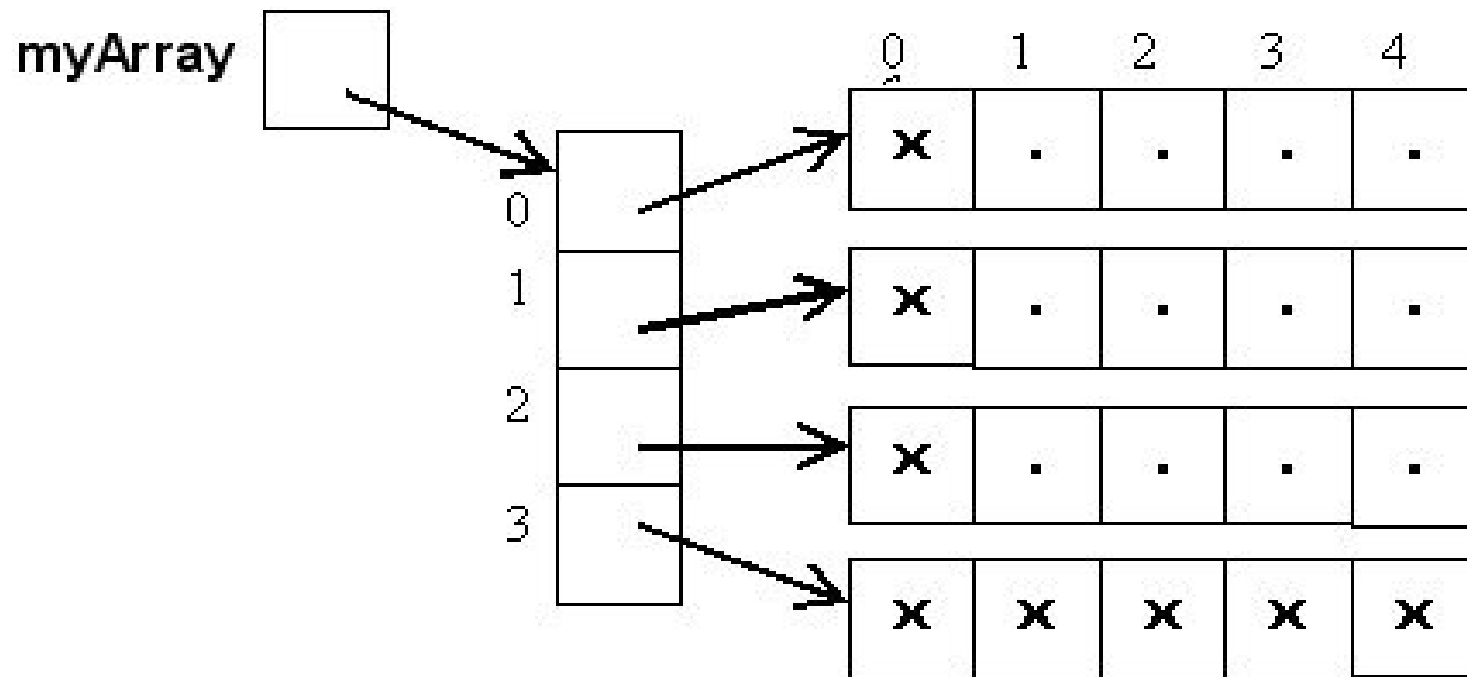
# Array

- `String[] names;`  
`names = new String[3];`  
`names[0] = "Georgianna";`  
`names[1] = "Jen";`  
`names[2] = "Simon";`
- `String[] names = {`  
`"Georgianna",`  
`"Jen",`  
`"Simon"`  
`};`

# Array

- `MyDate[] dates;`  
`dates = new MyDate[3];`  
`dates[0] = new MyDate(22, 7, 1964);`  
`dates[1] = new MyDate(1, 1, 2000);`  
`dates[2] = new MyDate(22, 12, 1964);`
- `MyDate[] dates = {`  
`new MyDate(22, 7, 1964),`  
`new MyDate(1, 1, 2000),`  
`new MyDate(22, 12, 1964)`  
`};`

# Multi Dimensional Array



# Multi Dimensional Array



## *Arrays of arrays:*

- `int[][] twoDim = new int[4][];`
- `twoDim[0] = new int[5];`
- `twoDim[1] = new int[5];`
- `int[][] twoDim = new int[][4]; // illegal`

# Multi Dimensional Array



*Array of four arrays of five integers each:*

- `int[][] twoDim = new int[4][5];`



# Multi Dimensional Array



*All array subscripts begin at 0:*

- ```
public void printElements(int[] list) {  
    for (int i = 0; i < list.length; i++) {  
        System.out.println(list[i]);  
    }  
}
```

# Array

- You cannot resize an array.
- You can use the same reference variable to refer to an entirely new array, such as:

```
int[] myArray = new int[6];
```

```
myArray = new int[10];
```

# Array

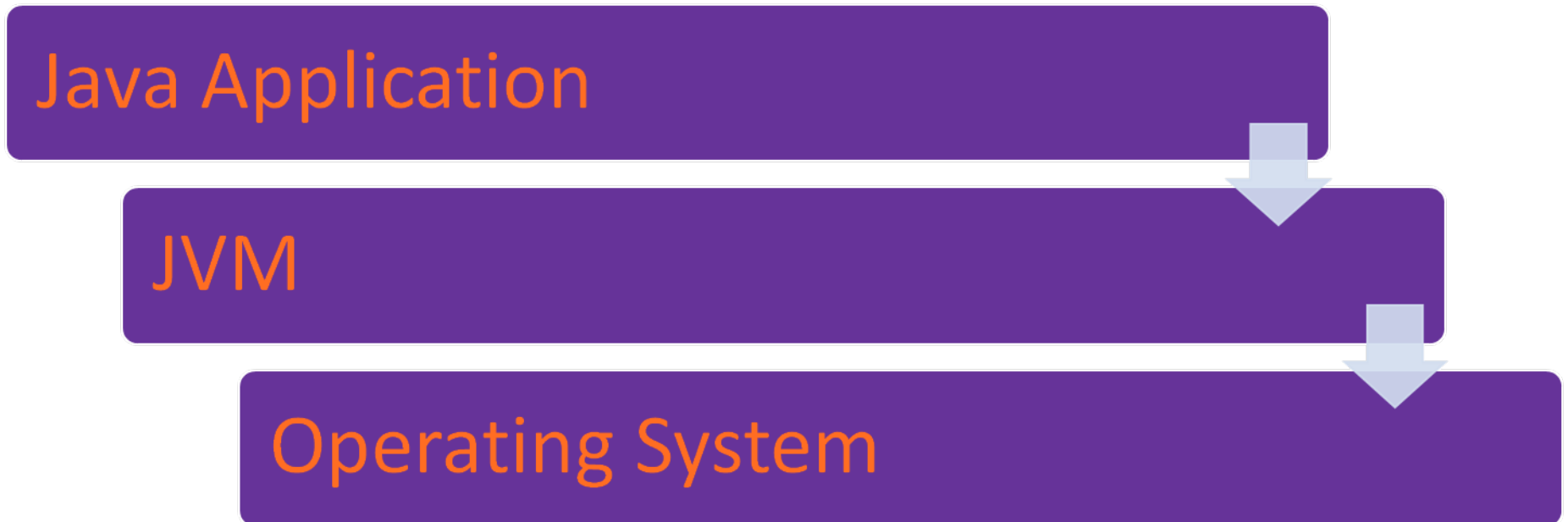
## *Copying Arrays:*

- The `System.arraycopy()` method to copy arrays is:
- `//original array`
- `int[] myArray = { 1, 2, 3, 4, 5, 6 };`
- `// new larger array`
- `int[] hold = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };`
- `/ copy all of the myArray array to the hold`
- `// array, starting with the 0th index`
- `System.arraycopy(myArray, 0, hold, 0, myArray.length);`

# JVM Architecture



# The JVM Architecture

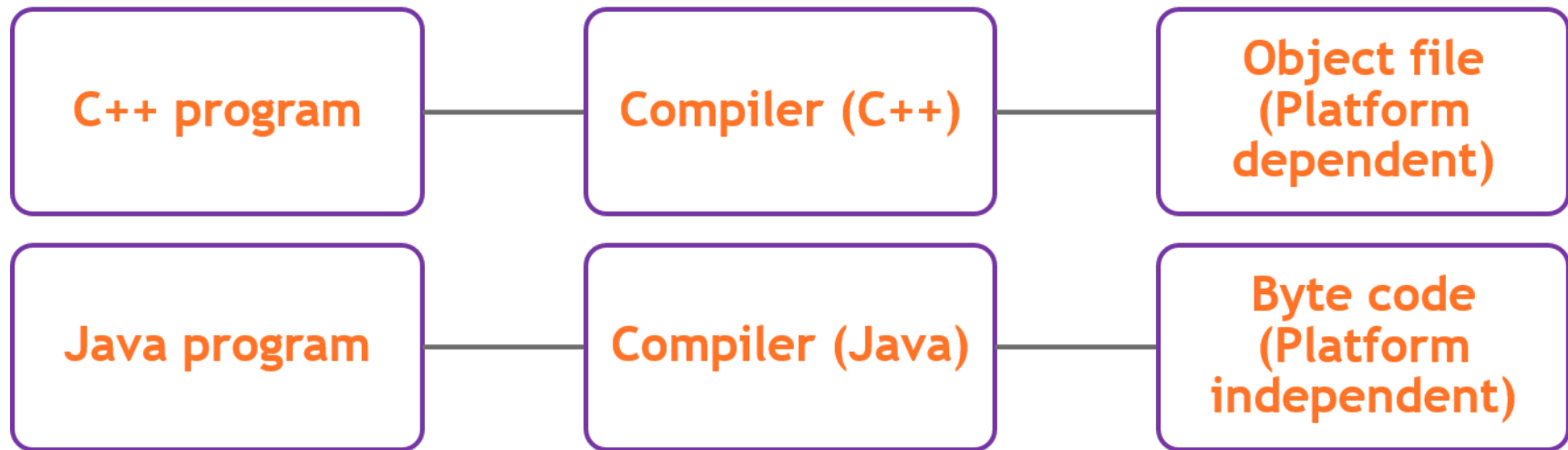


# The JVM Architecture



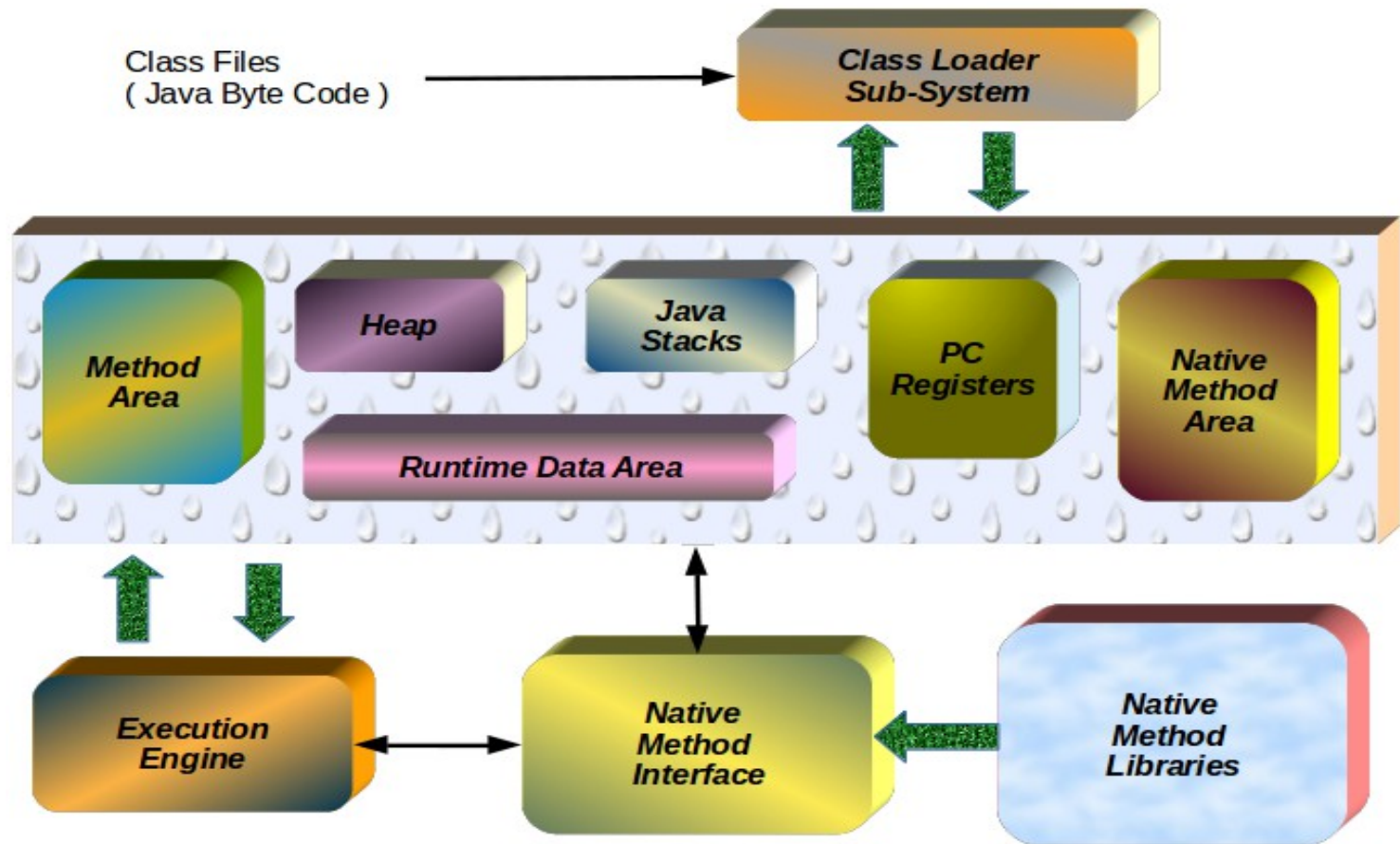
- JVM is Java Virtual Machine to execute Java programs.
- JVM and Byte Code make Java programs platform independent.

# Comparing C++ and Java



# JVM Architecture

## Deep Dive





# JVM Architecture Detail



*Class loader subsystem: When the JVM is started, three class loader are used:*

- System class loader: System class loader maps the class-path environment variables to load the byte code.
- Extension class loader: Extension class loader loads the byte code from jre/lib/ext.
- Bootstrap class loader: Bootstrap class loader loads the byte code from jre/lib.

# JVM Architecture Detail



*Method area: Method area (Class area) stores the structure of the class once it is loaded by the class loader. Method area is very important, it does two things once the class is stored in this area:*

- Identification and
- Execution

# JVM Architecture Detail



- Identification: All static members (variable, block, method, etc) are identified from top to bottom.
- Execution: Static variables and static blocks are executed after identification phase from top to bottom and static methods are executed when they are called. Once all static variables and blocks are executed then only static method will be executed and other static method will be executed when they are called.

***Write a program to implement identification and execution phase of method area.***

# JVM Architecture Detail



*Heap: Heap area stores the object. Objects instances are created at this area. When a class is having instance members (instance variable, instance method and instance block) then these members are identified and executed only when the instance is created at heap area.*

***Write a program to explain how instance member get identified and executed?***

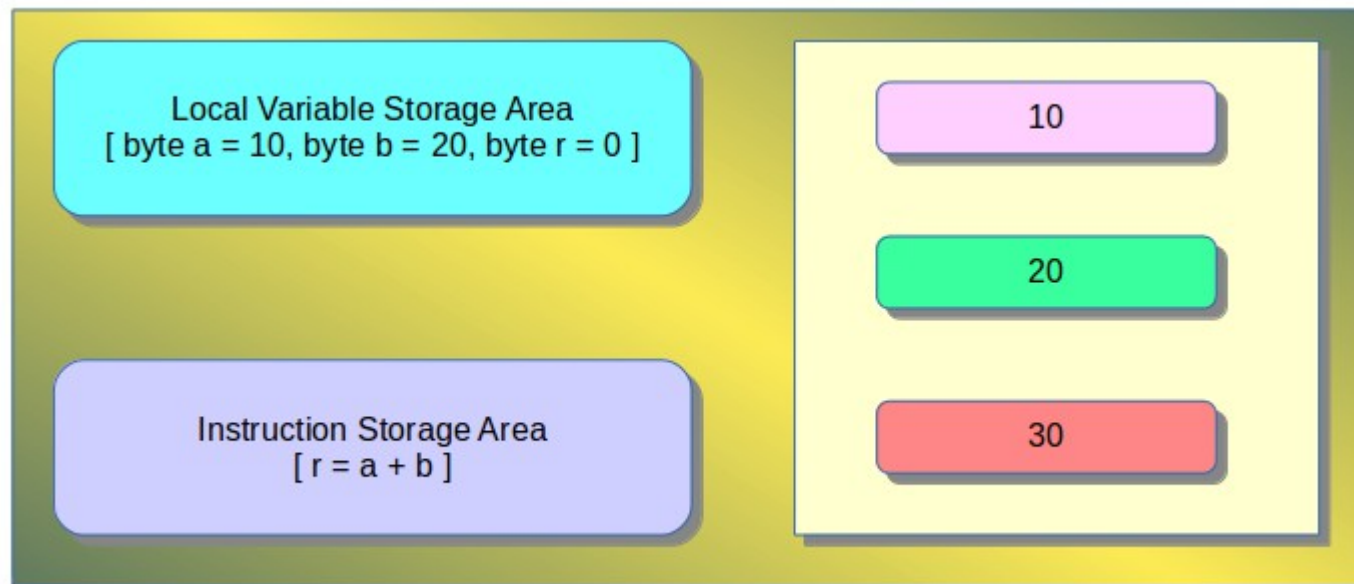
# JVM Architecture Detail



## *Java stacks:*

- In Java stack area two threads (main thread and garbage collector) are always running. When user create any new thread then it becomes third thread (Thread-0).
- When user creates any methods, it is executed by main thread, inside a stack frame. Each method gets their own stack frame to execute.
- Stack frame has three section, Local variable storage section, instruction storage section and memory slots to perform operations.
- Each memory slots inside stack frame is by default of 4 bytes but according to the size of variable, the size of the slot also gets shrink or expanded.

# JVM Architecture Detail



# JVM Architecture Detail



- *PC register:*

Program Counter (PC) register contains the address of the Java virtual machine instruction currently being executed.

- *Native method stacks:*

All the native methods are executed in this area.

# JVM Architecture Detail



- *Execution engine:*

All executions happening in JVM are controlled by Execution Engine.

- *Native method interface:*

Java Native Interface (JNI) enables the Java code running in Java Virtual Machine to call and be called by native application and libraries (Native Method Libraries) written in other language such as C and C++.



# JVM: Question



- What is the order of execution when all the member of static in a class?
- Explain the order of execution when class members are static and non static?
- What is the order of execution in case of inheritance?

# Stay connected

**About us:** Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,

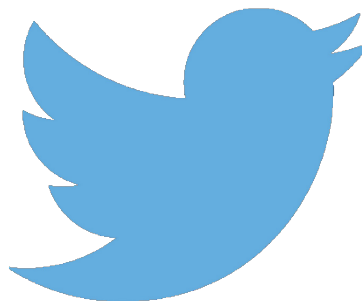
No-1, 9th Cross, 5th Main,  
Jayamahal Extension,  
Bangalore, Karnataka 560046

T: +91 80 6562 9666

E: [training@emertxe.com](mailto:training@emertxe.com)



<https://www.facebook.com/Emertxe>



<https://twitter.com/EmertxeTweet>



<https://www.slideshare.net/EmertxeSlides>

THANK YOU