# Java Basics

Java is a versatile programming language used to build applications for various devices. It's known for its "write once, run anywhere" ability and is popular for enterprise software. Java is simple to learn, robust, and secure. Java was developed by James Gosling at sun microsystems inc. in May 1995 and later acquired by oracle Corporation.

- Used to develop mobile apps, desktop apps, web apps, web servers, games, and enterprise-level systems.
- Popular platforms like LinkedIn, Amazon, and Netflix rely on Java for their back-end architecture, showcasing its stability and scalability across different environments.
- Popularity is so high that 3 billion+ devices use Java across the world.

**Java History:**

Java's history is as interesting as it is impactful.

James Gosling (Create one language) --- > "OAK" ---> Java Coffee change Java Green Team (Sun Microsystem emp)

| | | |
|---|---|---|
| ---> 1st Realse 1996 | --- Java 1.0 | For Public |
| ---> 1997 | --- Java ISO | implementation at no cost, JE System |
| ---> Number 13, 2006 | --- JVM | Free, Open-source software. |
| ---> May 8, 2006 | --- Full JVM | Fully available under open-source |

Java was designed with core principles: simplicity, robustness, security, high performance, portability, multi-threading, and dynamic interpretation. These principles have made Java a preferred language for various applications, including mobile devices, internet programming, gaming, and e-business.

Today, Java continues to be a cornerstone of modern software development, widely used across industries and platforms.

**Key Features:**

- Platform Independent:
  Compiler converts source code to byte code and then the JVM executes the bytecode generated by the compiler. This byte code can run on any platform.

- Simplicity

  Java's syntax is simple and easy to learn, especially for those familiar with C or C++. It eliminates complex features like pointers and multiple inheritances, making it easier to write, debug, and maintain code.

- Robustness

  Java language is robust which means reliable. It is developed in such a way that it puts a lot of effort into checking errors as early as possible, that is why the java compiler is able to detect even those errors that are not easy to detect by another programming language. The main features of java that make it robust are garbage collection, exception handling, and memory allocation.

- Security

  In java, we don't have pointers, so we cannot access out-of-bound arrays i.e it shows ArrayIndexOutOfBound Exception if we try to do so. That's why several security flaws like stack corruption or buffer overflow are impossible to exploit in Java. Also, java programs run in an environment that is independent of the os(operating system) environment which makes java programs more secure.

- Distributed

  We can create distributed applications using the java programming language. Remote Method Invocation and Enterprise Java Beans are used for creating distributed applications in java. The java programs can be easily distributed on one or more systems that are connected to each other through an internet connection.

- Multithreading

  As we know, java code written on one machine can be run on another machine. The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable.

- High Performance

  Java architecture is defined in such a way that it reduces overhead during the runtime and at sometimes java uses Just in Time (JIT) compiler where the compiler compiles code on-demand basis where it only compiles those methods that are called making applications to execute faster.

Java Code Executes

The execution of a Java application code involves three main steps:

- Creating the Program save .java ------ Source file
- Compiling the Program bytecode .class ------ Byte file
- Running the Program JVM executes ------ Machine code

## Essential Java Terminologies You Need to Know

- Java Virtual Machine (JVM)
  The JVM is an integral part of the Java platform, responsible for executing Java bytecode. It ensures that the output of Java programs is consistent across different platforms.

  1. The compilation is done by the JAVAC compiler which is a primary Java compiler included in the Java development kit (JDK). It takes the Java program as input and generates bytecode as output.

  2. In the Running phase of a program, JVM executes the bytecode generated by the compiler.

- Bytecode
  Bytecode is the intermediate representation of Java code, generated by the Java compiler. It is platform-independent and can be executed by the JVM.

- Java Development Kit (JDK)
  JDK is a complete Java development kit that includes everything including compiler, Java Runtime Environment (JRE), Java Debuggers, Java Docs, etc.

- Java Runtime Environment (JRE)
  JDK includes JRE. JRE installation on our computers allows the java program to run, however, we cannot compile it. JRE includes a browser, JVM, applet support, and plugins. For running the java program, a computer needs JRE.

- Garbage Collector
  In Java, programmers can't delete the objects. To delete or recollect that memory JVM has a program called Garbage Collector. Garbage Collectors can recollect the objects that are not referenced. So, Java makes the life of a programmer easy by handling memory management. However, programmers should be careful about their code whether they are using objects that have been used for a long time. Because Garbage cannot recover the memory of objects being referenced.

- Classpath
  Class path is the file path where the java runtime and Java compiler look for .class files to load. By default, JDK provides many libraries. If you want to include external libraries, they should be added to the class path.

### Advantages & Disadvantages

- Platform independent
- Object-Oriented programming
- Security
- Large community and Enterprise-level application

× Performance
× Memory management

**JDK vs JVM vs JRE**

Java Development Kit (JDK) is a software development environment used for developing java application and applets.

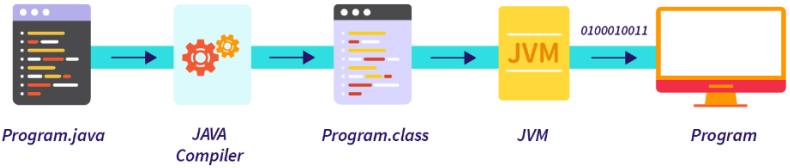
---> Java Source file
---> Compiled bytecode (JDK)
---> Executed code (JVM)

Java Runtime Environment (JRE) provides an environment to run only the java program onto the system.

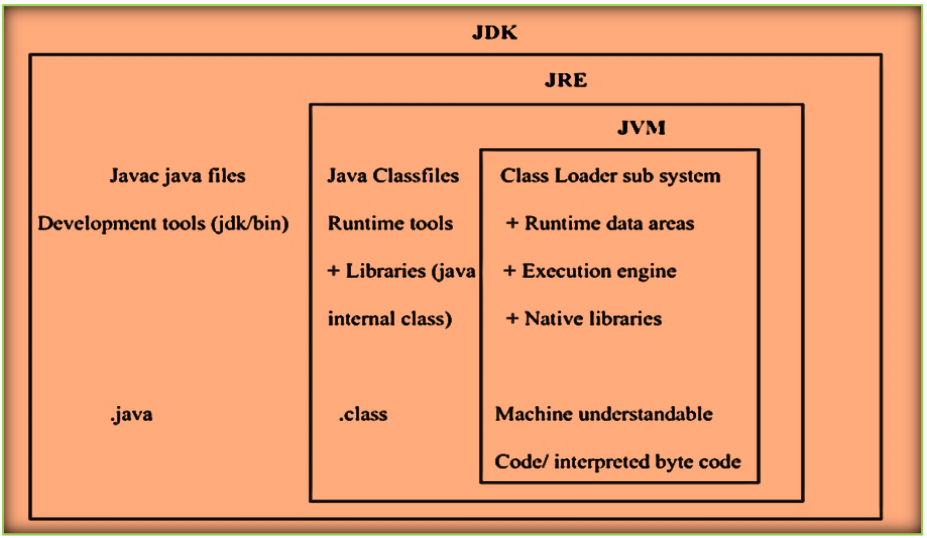
---> .class load class loader --- Bytecode verifier --- JVM interprets --- Exection

Java Virtual Machine (JVM) is responsible for executing the java program, need JDK-JRE.

---> Loading
---> Linking
---> Initialization




**Until java8** → JDK & JRE both are separately installed After java 8 combine.




@ayekiran

## Class Loader sub system

@ Loding

-- Bootstrap class loader rt.jar file / class from rt.jar loads.

-- Extension class loader jdk / jre / lib / ext ...jar files loads.

-- Application class loader Excel file (no interal support) → jexcel, Apache POI

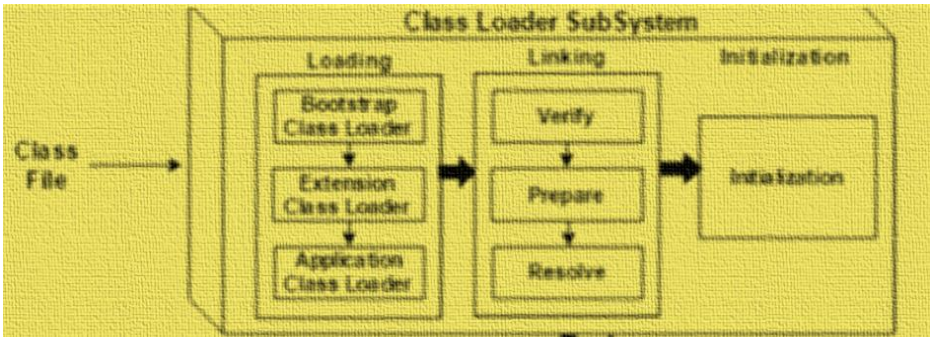
@ Linking

-- Verify face javap -v file. Class (byte code verify) → verify exception

-- Prepare face static variable initialization (default value)

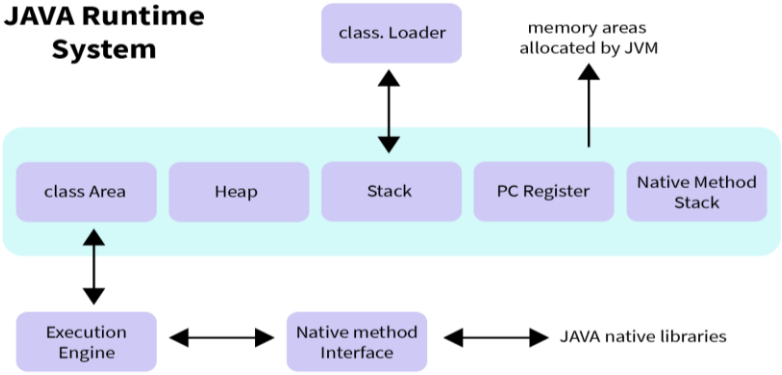
-- Resolve face Symbolling reference change to method reference (const pool)

@ Initialization




## Differences between JDK, JRE and JVM:

| JDK | JRE | JVM |
|---|---|---|
| **Used to develop Java applications.** | Used to run Java applications | Responsible for running Java code. |
| **Platform-dependent** | Platform-dependent | Platform-independent |
| **It includes development tools like (compiler) + JRE** | It includes libraries to run Java application + JVM | It runs the java byte code and make java application to work on any platform. |
| **Writing and compiling Java code.** | Running a Java application on a system. | Convert bytecode into native machine code. |




## Java Identifiers

An identifier is the name given to Variables, Classes, Methods, Packages, Interfaces, etc. These are the unique names and every Java Variables must be identified with unique names.

### Rules For Naming

- The only allowed characters for identifiers are all alphanumeric characters([A-Z],[a-z],[0-9]), '$'(dollar sign) and '_' (underscore).
- Identifiers should not start with digits ([0-9]).
- Java identifiers are case-sensitive.
- There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.
- Reserved Words can't be used as an identifier.

### Reserved Words

Any programming language reserves some words to represent functionalities defined by that language. Here:

1. keywords (50) define functionalities.
2. Literals (3) define value.

## Datatypes

A data type is a classification of data which tells the compiler or interpreter how the programmer intends to use the data because java is statically typed and also a strongly typed language.

| Primitive Data Type | Reference/object data types |
|---|---|
| 1. Byte | × String |
| 2. Short | × Class |
| 3. Int | × Object |
| 4. Long | × Interface |
| 5. Float | × Array |
| 6. double | |
| 7. char | |
| 8. boolean | |

Primitive data are only single values and have no special capabilities.

1. byte

   The byte data type is an 8-bit signed two's complement integer. The byte data type is useful for saving memory in large arrays.

   byte byteVar; --- > Size: 1 byte (8 bits) & 0 to 27-1

2. short

   The short data type is a 16-bit signed two's complement integer. Similar to byte.

   short shortVar; --- > Size: 2 bytes (16 bits) & 0 to 215-1

3. int

   It is a 32-bit signed two's complement integer.

   int intVar; --- > Size: 4 bytes (32 bits) & 0 to 232-1

4. long

   The long data type is a 64-bit signed two's complement integer. It is used when an int is not large enough to hold a value, offering a much broader range.

   long longVar; --- > Size: 8 bytes (64 bits) & 0 to 264-1

5. float

   The float data type is a single-precision 32-bit IEEE 754 floating-point. Use a float (instead of double) if you need to save memory in large arrays of floating-point numbers.

   float floatVar; --- > Size: 4 bytes (32 bits) & 0 to 3.4e38

6. double

   The double data type is a double-precision 64-bit IEEE 754 floating-point. For decimal values, this data type is generally the default choice.

   double doubleVar; --- > Size: 8 bytes (64 bits) & 0 to 1.7e130

Note: Both float and double data types were designed especially for scientific calculations, where approximation errors are acceptable. If accuracy is the most prior use BigDecimal class instead.

7. Char

The char data type is a single 16-bit Unicode character with the size of 2 bytes (16 bit).

char charVar; --- > Size: 2 bytes (16 bits) & ASCII (0-255)

8. Boolean

The boolean data type represents a logical value that can be either true or false.

boolean booleanVar; --- > Size: Virtual machine dependent (typically 1 byte, 8 bi)

**Type Conversion:**

Type conversion is the process of converting a variable from one datatype to another.

Implicit casting (Widening Conversion): Automatic conversion from a smaller to a larger

byte --- > short (or) char--- > int --- > long --- > float --- > double

For example: int i = 100;
long l = I;

Explicit casting (Narrowing Conversion): Manual conversion from a lager to a smaller

For example: double d = 10.5;
int i = (int) d;

Importance of Datatypes:

Understanding data types is crucial for effective programming as they define how define how data is manipulated, stored, and managed in the code.

---> Memory management
---> Type Safety
---> Code Clarity

**Variable**

Variables are the containers for storing the data values or you can also call it a memory location name for the data.

Variable assign 2 way:
@ Initialization (known at define time)
@ Define after Initialization

Variable Naming Rules:
@ Must begin with a letter (A-Z), dollar sign ($), or underscore (_ for MATH_PI)
@ Subsequent characters can be letters, digits (0-9), dollar sign ($), or underscore.
@ Case-sensitive (ex: age, AGE, Age are different variables)
@ Can't use java reserved keywords (ex: class, void, int, .....)

Variable types

1. Global Scope Variables

- Instance variables (non-static fields)

  Instance variables are non-static variables and are declared in a class outside of any method, constructor, or block.
  ---> Access only after object creation or this

- Class Variable (Static Fields): No need to reference

  Static variables are created when the program starts (when the class is loaded) and are destroyed when the program ends. This makes them part of the class's memory space, rather than tied to a particular object.

| Aspect | Instance Variables | Static Variables |
|---|---|---|
| **Definition** | Variables specific to an instance of a class. | **Variables shared by all instances of a class.** |
| **Memory Allocation** | Each object has its own copy of instance variables. | **Only one copy of the static variable is created, shared by all objects.** |
| **Scope** | Scope is limited to the specific object (instance). | **Scope is global to the class, accessible via any object of that class.** |
| **Access** | Accessed using an object of the class. | **Accessed using the class name or through objects.** |
| **Lifetime** | Lifetime is tied to the object. | **Lifetime is tied to the class, persists as long as the class is loaded.** |
| **Initialization** | Initialized when the object is created. | **Initialized once when the class is loaded into memory.** |

| | | |
|---|---|---|
| **Default Value** | Default value is null or zero (depends on the data type) unless explicitly initialized. | **Default value is null or zero (depends on the data type) unless explicitly initialized.** |
| **Modification** | Can be modified for each object. | **Modifying a static variable affects all instances of the class.** |
| **Memory Usage** | Consumes memory for every object created. | **Consumes memory only once, shared across all objects.** |
| **Usage Example** | **Used to store properties specific to an instance (e.g., name, age).** | **Used for class-wide data (e.g., a counter for instances).** |

2. Local Scope Variable (By default non-static Field)

- Local Variable: Access only local methods.
  A variable defined within a block or method or constructor is called a local variable.

- Parameters: We can't assign directly, only define

3. Final variable (Value can't be changed once it is initialized)

Best Practices
@ Choose meaningful variable names to make the code more readable.
@ Follow camelCase naming convention for variables.
@ Avoid using single-character variable names except for loop indices (e, g, i, j ...)

Variable use building blocks for storing and manipulating data throughout the lifecycle of a java application.

**Operators:**
operators are special symbols that perform operations on variables or values.


**Types of Operators**

- Arithmetic Operators:
  Perform mathematical operations like addition, subtraction, multiplication, division, and modulus (+, -, *, /, %).

- Unary Operators:
  Operate on a single operand to perform operations like increment (++), decrement (--), negation (-), and logical NOT (!).

- Assignment Operator:
  Assigns values to variables (=, +=, -=, *=, /=, etc.).

- Relational Operators:
  Compare two values and return a boolean result (==, !=, >, <, >=, <=).

- Logical Operators:
  Combine multiple boolean expressions (&& for AND, || for OR, ! for NOT).

- Ternary Operator:
  A shorthand for if-else statement, with syntax condition? true Val: false Val

- Bitwise Operators:
  Perform bit-level operations on integer values (&, |, ^, ~, <<, >>).

- Shift Operators:
  Shift bits left or right, altering the binary representation of values (<< for left shift, >> for right shift).

Instance of Operator:
Checks if an object is an instance of a specific class (instanceof in Java or isinstance () in Python).

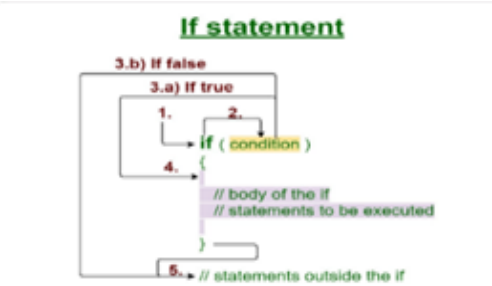
Understanding these operators and their correct usage is essential for writing effective and efficient java code.

**Control Statements:**
Control statements in programming are constructs that allow a program to make decisions or control the flow of execution based on certain conditions. They determine the sequence in which instructions are executed, enabling the program to respond differently under various circumstances.

**Conditional/Decision-making Statements:**
It allows the program to make decisions based on conditions.

1. if: Executes a block of code if a condition evaluates is true.

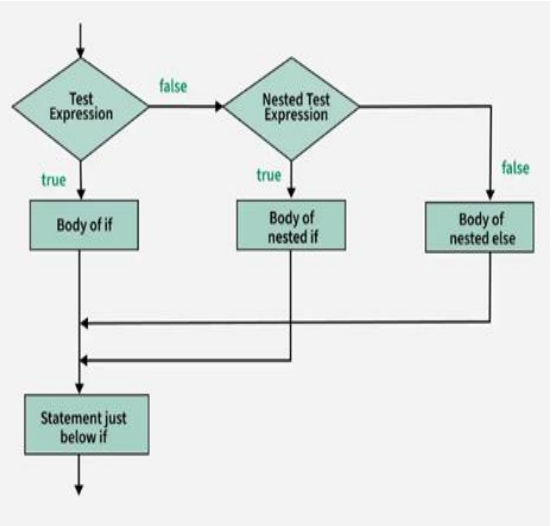
```
if(condition) {
    // Statements
  // condition true
else {
  // condition true
}
```




2. else: An alternative block of code if the condition is false.

3. else if: Provides an alternative condition to check if the first if condition is false.

```
if (condition1) {
  // code to be executed if
  condition1 is true
} else if (condition2) {
  // code to be executed if
  condition2 is true
} else {
  // code to be executed if
  all conditions are false
  }
```




4. nested if: Make a series of decisions based on multiple conditions.

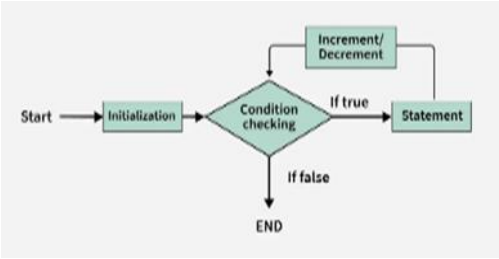
```
if (condition1) {
   if (condition2) {
      if (condition3) {
            // statements;
         }

      }
}
```

5. Switch statement: Execute parts of code based on the value of a variable.

```
switch (expression) {
  case value1:
   // code to be executed if
      expression == value1
    break;

  case value2:
    // code to be executed if  expression == value2
    break;
   // more cases...
  default:
```



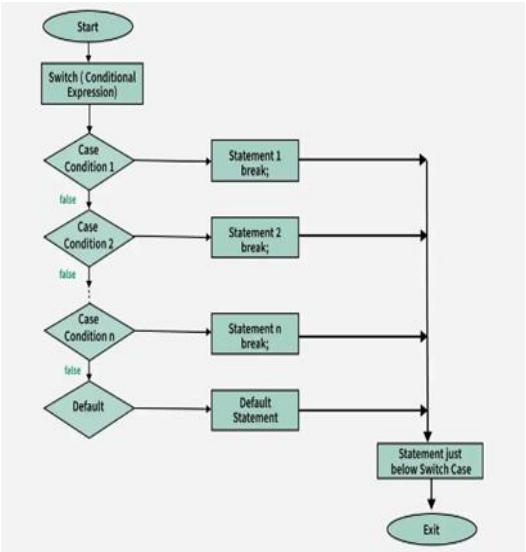

**Looping (Repetition) Statements:**
It allows a block of code to be executed repeatedly based on a condition.

1. for loop: Repeats a block of code a specific number of times or iterate.

```
for (initialization; condition;
        increment/decrement) {
  // code to be executed
}
```

2. Enhanced for loop (for each): Specifically used for iterating over arrays or collections

```
for (dataType variable : arrayOrCollection) {
   // code to be executed
}
```

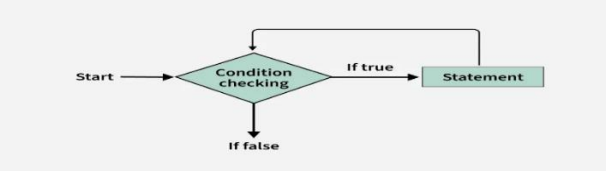


3. while loop: It is used when we want to check the condition before running the code.

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

4. do while loop: Similar to the while but guarantees that the code block executes at least once before the condition is checked.

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




**Control/Jump Statement:**

It controls the flow by jumping out of loops or skipping certain parts of code (or) controlling overall code.

1. break: Terminates/Exit the loop (or) switch statement it is in.

2. continue: Skip the current iteration of a loop and proce - eds with next iteration.

3. return: Exits from the current method & optionally returns a value.