

# Introduction To Java

---

# Why Java?

*"Today from web applications and desktop GUI's to the Internet of Things and self-driving cars, Java is everywhere."*

Java follows the principle of WORA (Write Once, Run Anywhere)

Version 1.0 rolled out in 1996 when Sun Microsystems promised the principle of WORA (Write Once, Run Anywhere).

Then came along Java 2 (J2SE 1.2) in December 1998-1999. J2EE was for enterprise applications.

Then in 2006, boosting its marketing capabilities, Sun renamed new J2xx versions as Java EE, Java ME, and Java SE.



**James Gosling,  
founder of Java**

## Where we see Java?

- Desktop applications
- Web applications
- Mobile applications (Android)
- Cloud computing
- Enterprise applications
- Scientific applications
- Operating Systems
- Embedded systems
- Real-time software
- Cryptography
- Smart cards
- Computer games
- Web Servers & Application servers

## JDK (Java Development Kit)

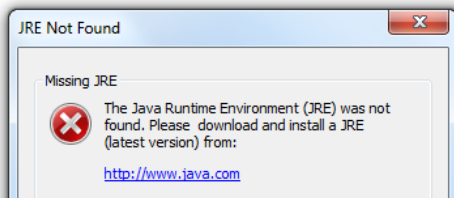
JDK contains everything that will be required to develop and run Java application.

## JRE (Java Runtime Environment)

JRE contains everything required to run Java application which has already been compiled. It doesn't contain the code library required to develop Java application.

## JVM (Java Virtual Machine)

JVM is a virtual machine which works on top of your operating system to provide a recommended environment for your compiled Java code. JVM only works with bytecode. Hence you need to compile your Java application(.java) so that it can be converted to bytecode format (also known as the .class file). Which then will be used by JVM to run an application. JVM only provide the environment needed to executed Java Bytecode.



## JDK - Java Development Kit

Developer <img alt="Developer icon" data-bbox="875 375 899 399"/>

### Development Tools

java, javac, JShell, javadoc, jar, jdeps, JConsole, VisualVM, JFR, JPDA, JVM TI, IDL, Java DB, Debugging Tools, Deployment Tools, Monitoring Tools

## JRE - Java Runtime Environment

Java App Users <img alt="Java App Users icon" data-bbox="840 522 861 553"/>

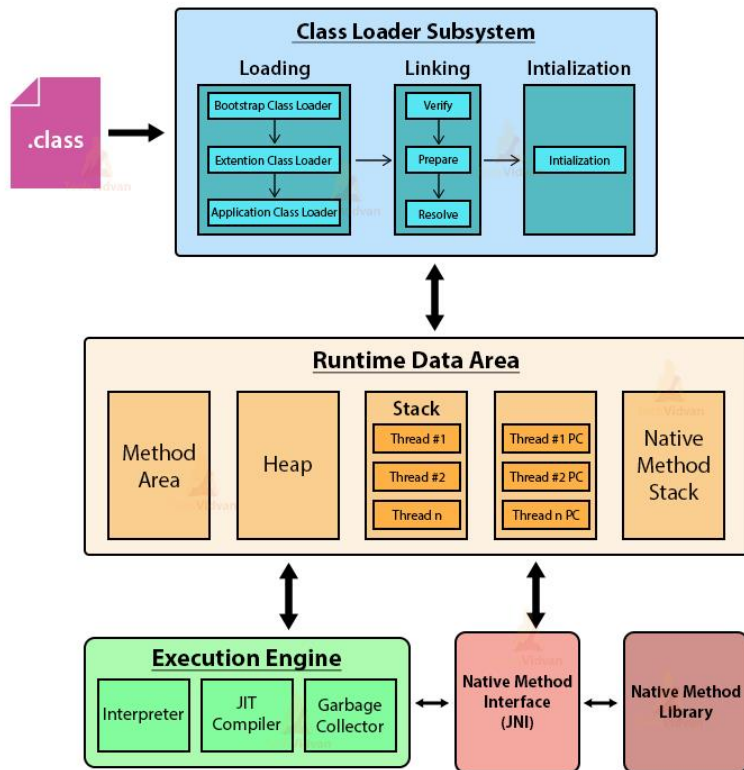
Java API, Runtime Libraries, Byte Code Verifier,  
Libraries: User Interface (Swing, JavaFx, Sound, Image I/O .....),  
Integration Libraries (JDBC, RMI, . . .), Serialization, Netowrking,  
lang and util libraries (Math, Collections, Concurrency, Logging, ...).....

## JVM - Java Virtual Machine

Java Interpreter, JIT, Garbage Collector, Thread Synchronization,  
Class Loader Subsystem

JDK = JRE + Development Tools  
JRE= JVM + Libraries

## JVM Model



### JVM Memory Area

1. **Method Area** – It stores the structure of each class like method data, field data, runtime pool, metadata.
2. **Heap**– Heap is the runtime area where object allocation takes place.
3. **Stacks**– Stacks store the partial results and local variables of a program. Whenever a thread is created, there is a simultaneous creation of JVM stack. When we invoke a method, a new frame creates and destroys at the same time when the invocation process completes.

**JVM is platform dependent because it takes java byte code and generates byte code for the current operating system.**

# What is OpenJDK vs AdoptOpenJDK

OpenJDK is an open-source project providing source-code (not builds) of an implementation of the Java platform

It is the result of an effort Sun Microsystems began in 2006.

OpenJDK is the official reference implementation of Java SE since version 7.

AdoptOpenJDK is an organization founded by some prominent members of the Java community aimed at providing binary builds and installers at no cost for users of Java technology.

Build	Organization	LTS	Permissive license	TCK tested	Built unmodified	Commercial support
AdoptOpenJDK <sup>[24]</sup> (moved to Eclipse as Eclipse Adoptium in 2021) <sup>[25]</sup>		Yes	Yes	No	Optional	Optional (IBM)
Alibaba Dragonwell <sup>[26]</sup>	Alibaba Group	Yes	Yes	No	No	No
Amazon Corretto <sup>[27]</sup>	Amazon	Yes	Yes	Yes	No <sup>[28]</sup>	Optional (on AWS)
Azul Zulu <sup>[29]</sup>	Azul Systems	Yes	Yes	Yes	No	Optional
BellSoft Liberica JDK <sup>[30]</sup>	BellSoft	Yes	Yes	Yes	No	Optional
Eclipse Adoptium/Temurin <sup>[31]</sup>	Eclipse Foundation	Yes	Yes	Yes	No	Optional (Azul, IBM)
IBM Semeru Runtime Certified Edition <sup>[32]</sup>	IBM	Yes	No <sup>[33]</sup>	Yes	No	Optional (IBM)
IBM Semeru Runtime Open Edition <sup>[34]</sup>	IBM	Yes	Yes <sup>[33]</sup>	No	No	Optional (IBM)
IBM Java SDK <sup>[35]</sup> (version 11 moved to IBM Semeru Runtime Certified Edition)	IBM	Yes	No	Yes	No	Yes
JetBrains Runtime <sup>[36]</sup>	JetBrains	Yes	Yes	No	No	No
Microsoft Build of OpenJDK <sup>[37]</sup>	Microsoft	Yes	Yes	Yes	No	Optional (on Azure)
ojdkbuild <sup>[38]</sup>		Yes	Yes	No	Yes	No
OpenLogic OpenJDK <sup>[39]</sup>	OpenLogic	Yes	Yes	No	No	Optional
GraalVM Community Edition <sup>[40]</sup>	GraalVM	No	Yes	Yes	No	No
Oracle GraalVM Enterprise Edition <sup>[41]</sup>	Oracle Corporation	Yes	No	Yes	No	Yes
Oracle Java SE <sup>[42]</sup>	Oracle Corporation	Yes	No	Yes	No	Yes
Oracle OpenJDK <sup>[43]</sup>	Oracle Corporation	No	Yes	Yes	No <sup>[44][45]</sup>	No
Red Hat build of OpenJDK <sup>[46]</sup>	Red Hat	Yes	Yes	Yes	No	Yes
SAP SapMachine <sup>[47]</sup>	SAP	Yes	Yes	Yes	No	Optional (for SAP products)

**OpenJDK → source code**  
**Adoptium/AdoptOpenJDK → builds**



Java Implementation  
is provided by may  
enterprises

## Download Java from where?

1. <https://adoptopenjdk.net/?variant=openjdk8&jvmVariant=hotspot> (OLD)
2. <https://adoptium.net/releases.html?variant=openjdk8> (NEW)

Note: AdoptOpenJDK is moving to the Eclipse Foundation and rebranding, future releases will come from Adoptium.net

Github Repo: [https://github.com/adoptium/temurin17-binaries/releases/download/jdk-17.0.2+8/OpenJDK17U-jdk\\_x64\\_windows\\_hotspot\\_17.0.2\\_8.msi](https://github.com/adoptium/temurin17-binaries/releases/download/jdk-17.0.2+8/OpenJDK17U-jdk_x64_windows_hotspot_17.0.2_8.msi)

Name	Date modified	Type	Size
▼ Today (2)			
 OpenJDK11U-jdk_x64_windows_hotspot_11.0.14_9	2/7/2022 2:16 AM	Windows Installer Pa...	171,088 KB
 OpenJDK8U-jdk_x64_windows_hotspot_8u322b06	2/7/2022 2:15 AM	Windows Installer Pa...	88,034 KB

<https://openjdk.org/projects/jdk/20/>

What are JEP in Openjdk?

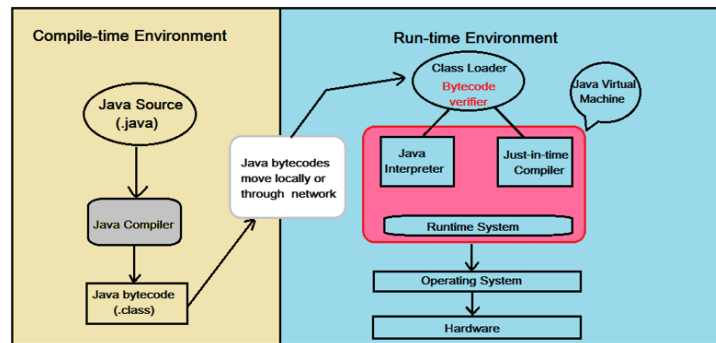
This JEP is the index of all JDK Enhancement Proposals, known as JEPs.

# Is Java a Compiled or an Interpreted programming language?

## Java does both compilation and interpretation

- In Java, programs are not compiled into executable files; they are compiled into bytecode (as discussed earlier), which the JVM (Java Virtual Machine) then interprets / executes at runtime. Java source code is compiled into bytecode when we use the javac compiler. The bytecode gets saved on the disk with the file extension .class.
- When the program is to be run, the bytecode may be converted, using the just-in-time (JIT) compiler. The result is machine code which is then fed to the memory and is executed.**
- Javac is the Java Compiler which Compiles Java code into Bytecode. JVM is Java Virtual Machine which Runs/ Interprets/ translates Bytecode into Native Machine Code. In Java though it is considered as an interpreted language, It may use JIT (Just-in-Time) compilation when the bytecode is in the JVM. The JIT compiler reads the bytecodes in many sections (or in full, rarely) and compiles them dynamically into machine code so the program can run faster, and then cached and reused later without needing to be recompiled. So JIT compilation combines the speed of compiled code with the flexibility of interpretation.
- An interpreted language** is a type of programming language for which most of its implementations execute instructions directly and freely, without previously compiling a program into machine-language instructions. The interpreter executes the program directly, translating each statement into a sequence of one or more subroutines already compiled into machine code.
- A compiled language** is a programming language whose implementations are typically compilers (translators that generate machine code from source code), and not interpreters (step-by-step executors of source code, where no pre-runtime translation takes place)
- In modern programming language implementations like in Java, it is increasingly popular for a platform to provide both options.

*JIT is optional and depends on the Java Implementation Providers.*



*The Java specs define bytecode.*

*Whether that bytecode is run*

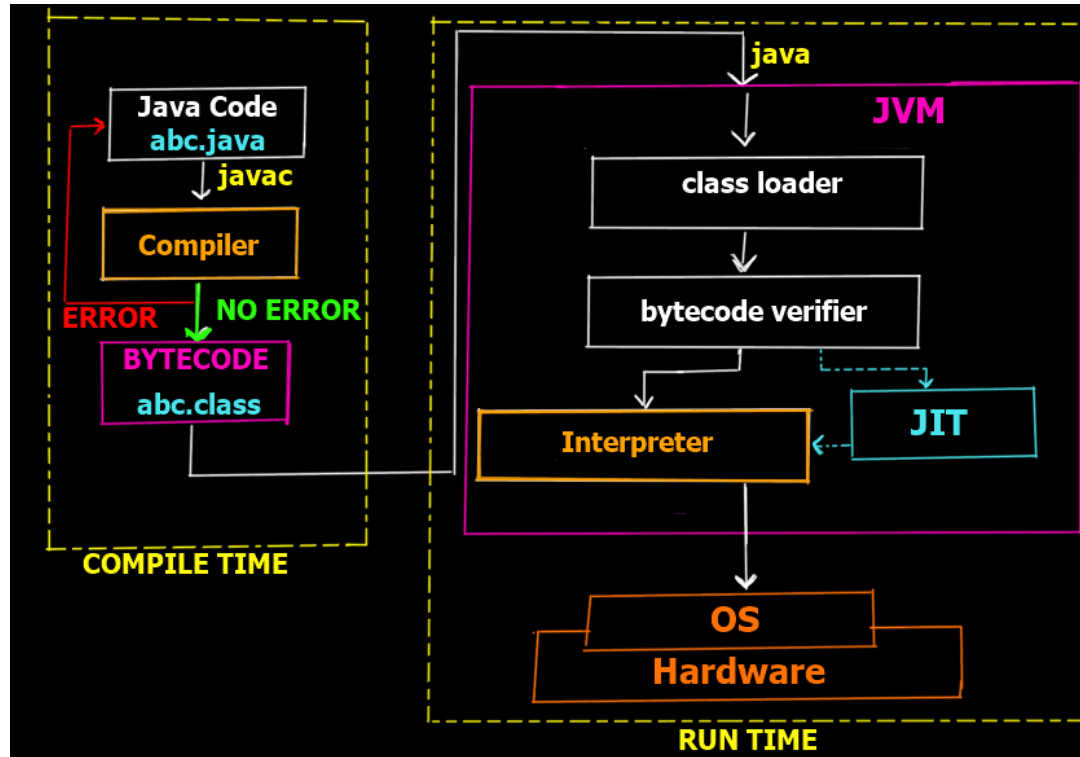
- (a) directly in hardware,
- (b) *through an interpreter*,
- (c) *compiled beforehand*, or
- (d) partially compiled on-the-fly at runtime are all left as implementation details.

*Note that all four of those options have indeed been used by various real-world Java implementations.*

*Modern JVMs use a technique called Just-in-Time (JIT) compilation to compile the bytecode to native instructions understood by hardware CPU on the fly at runtime.*

*Some implementations of JVM may choose to interpret the bytecode instead of JIT compiling it to machine code, and running it directly. While this is still considered an "interpreter".*

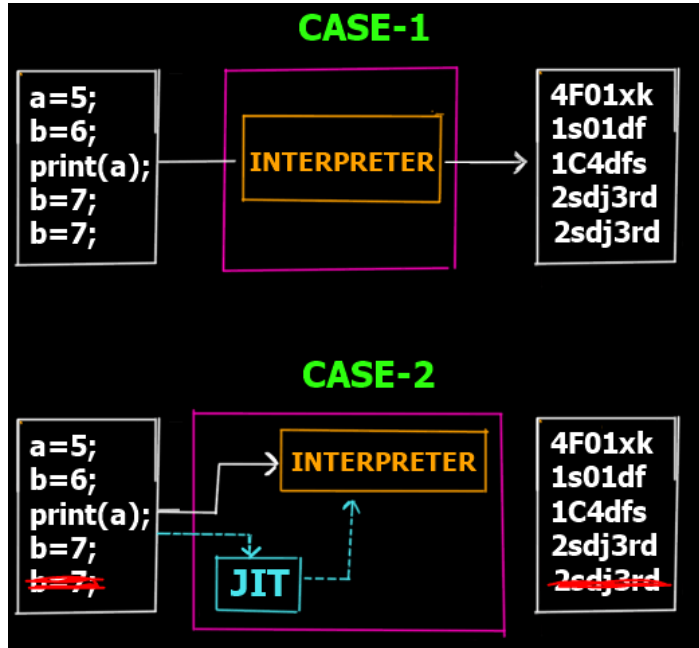
# Execution Process of Java Program



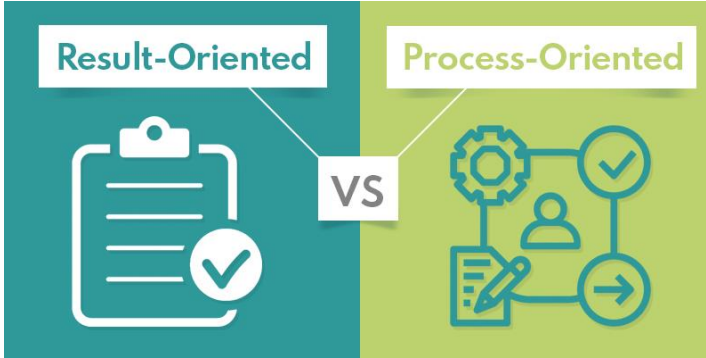
1. The compiler checks the code for syntax errors and any other compile time errors and if no error is found the compiler converts the java code into an intermediate code(abc.class file) known as **bytecode**.
2. This **intermediate code is platform independent** (you can take this bytecode from a machine running windows and use it in any other machine running Linux or MacOS etc).
3. Also this bytecode is an intermediate code, hence it is only understandable by the JVM and not the user or even the hardware /OS layer.
4. the bytecode is loaded into the JVM by the **class loader**(another inbuilt program inside the JVM)
5. Now the **bytecode verifier**(an inbuilt program inside the JVM) checks the bytecode for its integrity and if not issues are found passes it to the interpreter.
6. Since java is both compiled and interpreted language, now the interpreter inside the JVM converts each line of the bytecode into executable machine code and passed it to the OS/Hardware i.e. the CPU to execute.



# Working of JIT (Just-In-Time) Compiler (Just Improve Performance)



1. Let's assume we have 5 lines which are supposed to be interpreted to their corresponding machine code lines.
2. So as you can see in the Case 1 there is no JIT involved, thus the interpreter converts each line into its corresponding machine code line.
3. However if you notice the last 2 lines are the same (consider it a redundant line inserted by mistake).
4. Clearly that line is redundant and does not have any effect on the actual output but yet since the interpreter works line by line it still creates 5 lines of machine code for 5 lines of the bytecode.
5. In case 2 we have the JIT compiler. Now before the bytecode is passed onto the interpreter for conversion to machine code, the JIT compiler scans the full code to see if it can be optimized.
6. As it finds the last line is redundant it removes it from the bytecode and passes only 4 lines to the interpreter thus making it more efficient and faster as the interpreter now has 1 line less to interpret.
7. So this is how JIT compiler speeds up the overall execution process.
8. This was just one scenario where JIT compiler can help in making the execution process fast and efficient.
9. There are other cases like inclusion of only those packages needed in the code, code optimizations, redundant code removal etc which overall makes the process very fast and efficient.
10. Also different JITs developed by different companies work differently and JIT compilers are an optional step and not invoked every time (it can be disabled).

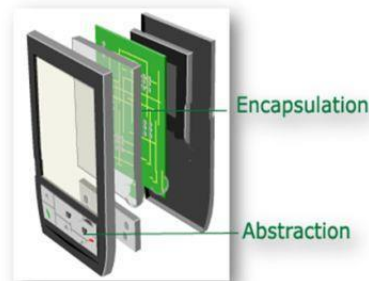


# Pillars of OOP Abstraction & Encapsulation

Abstraction is generalized term.

Encapsulation is subset of Abstraction.

Abstraction	Encapsulation
It solves an issue at the design level.	Encapsulation solves an issue at implementation level.
hides the unnecessary detail but shows the essential information.	It hides the code and data into a single entity or unit so that the data can be protected from the outside world.
Focuses on the external lookout.	Focuses on internal working.
Lets focus on what an object does instead of how it does it.	Lets focus on how an object does something.
Example: Outer look of mobile, like it has a display screen and buttons.	Example: Inner details of mobile, how button and display screen connect with each other using circuits.



Complex logic is in the circuit board which is encapsulated in a touchpad and a nice interface(buttons) is provided to abstract it out to the user.

Example: The **solution architect** is the person who creates the high-level **abstract** technical design of the entire solution, and this design is then handed over to the **development team** for **implementation**.

Here, solution architect acts as a abstract and development team acts as a Encapsulation.

## Pillars of OOP Abstraction & Encapsulation

Abstraction	Encapsulation
Abstraction is a general concept formed by extracting common features from specific examples or The act of withdrawing or removing something <b>unnecessary</b> .	Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both <b>safe from outside interference</b> and <b>misuse</b> .
You can use abstraction using <b>Interface</b> and <b>Abstract</b> Class	You can implement encapsulation using <b>Access Modifiers</b> (Public, Protected & Private)
Abstraction solves the problem in <b>Design</b> Level	Encapsulation solves the problem in <b>Implementation</b> Level
For simplicity, abstraction means hiding implementation using Abstract class and Interface	For simplicity, encapsulation means hiding data using getters and setters

# Pillars of OOP Abstraction & Encapsulation

## Abstraction:

- Abstraction is "To represent the essential feature without representing the background details."
- Abstraction lets you focus on what the object does instead of how it does it.
- Abstraction provides you a generalized view of your classes or object by providing relevant information.
- Abstraction is the process of hiding the working style of an object and showing the information of an object in understandable manner.

## Real world Example of Abstraction: -

- Suppose you have an object Mobile Phone.
- Suppose you have 3 mobile phones as following:-
  - Nokia 1400 (Features:- Calling, SMS)
  - Nokia 2700 (Features:- Calling, SMS, FM Radio, MP3, Camera)
  - Black Berry (Features:-Calling, SMS, FM Radio, MP3, Camera, Video Recording, Reading E-mails)
- Abstract information (Necessary and Common Information) for the object "Mobile Phone" is make a call to any number and can send SMS." so that, for mobile phone object you will have abstract class like code here-
- Abstraction means putting all the variables and methods in a class which are necessary. E.g. Abstract class and abstract method.
- Abstraction is the common thing.e.g. If somebody in your collage tell you to fill application form, you will fill your details like name, address, data of birth, which semester, percentage you have got etc. If some doctor gives you an application to fill the details, you will fill the details like name, address, date of birth, blood group, height and weight.
- See in the example what is the common thing?
- Age, name, address so you can create the class which consist of common thing that is called abstract class.
- That class is not complete, and it can inherit by other class.

```
abstract class MobilePhone
{
    public void calling();
    public void sendSMS();
}

public class Nokia1400 extends MobilePhone
{
}

public class Nokia2700 extends MobilePhone
{
    public void fmRadio(){//..}
    public void mp3() {//..}
    public void camera() {//..}
}

public class BlackBerry extends MobilePhone
{
    public void fmRadio() {//..}
    public void mp3() {//..}
    public void camera() {//..}
    public void recording() {//..}
    public void readAndSendEmails() {//..}
}
```

# Pillars of OOP Abstraction & Encapsulation

## Encapsulation:

- Wrapping up data member and method together into a single unit (i.e., Class) is called Encapsulation.
- Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.
- Encapsulation is like your bag in which you can keep your pen, book etc. It means this is the property of encapsulating members and functions.
- Encapsulation means hiding the internal details of an object, i.e., how an object does something.
- Encapsulation prevents clients from seeing its inside view, where the behavior of the abstraction is implemented.
- Encapsulation is a technique used to protect the information in an object from the other object.
- Hide the data for security such as making the variables as private and expose the property to access the private data which would be public.
- So, when you access the property, you can validate the data and set it.

### Real world Example of Encapsulation:-

- Let's take example of Mobile Phone and Mobile Phone Manufacturer
- Suppose you are a Mobile Phone Manufacturer and you designed and developed a Mobile Phone design(class), now by using machinery you are manufacturing a Mobile Phone(object) for selling, when you sell your Mobile Phone the user only learn how to use the Mobile Phone but not that how this Mobile Phone works.
- This means that you are creating the class with function and by making object (capsule) of it you are making availability of the functionality of you class by that object and without the interference in the original class.

### Example-2:

- TV operation, It is encapsulated with cover, and we can operate with remote and no need to open TV and change the channel.
- Here everything is in private except remote so that anyone can access not to operate and change the things in TV.

```
public class Bag
{
    book;
    pen;
    readBook();
}

public class Demo
{
    private int mark;

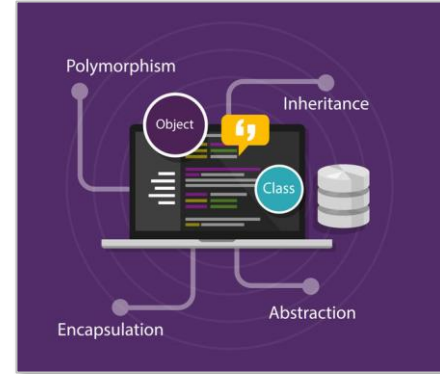
    public int getMark()
    { return mark; }

    public void setMark(int value)
    { if (value > 0)
        mark = value;
      else
        mark = 0;
    }
}
```

- **Encapsulation:**
  - The wrapping up of data and functions into a single unit (called class) is known as encapsulation.
  - Encapsulation containing and hiding information about an object, such as internal data structures and code.
  - Encapsulation is -
    - ✓ Hiding Complexity,
    - ✓ Binding Data and Function together,
    - ✓ Making Complicated Method's Private,
    - ✓ Making Instance Variable's Private,
    - ✓ Hiding Unnecessary Data and Functions from End User.
- **Encapsulation implements Abstraction.**
  - Abstraction is -
    - ✓ Showing what's Necessary,
    - ✓ Data needs to abstract from End User.

# Pillar of OOP Inheritance & Polymorphism

- **Inheritance** : The ability of creating a new class from an existing class. Inheritance is when an object acquires the property of another object. Inheritance allows a class (subclass) to acquire the properties and behavior of another class (super-class). It helps to reuse, customize and enhance the existing code. So it helps to write a code accurately and reduce the development time. When a class acquire the property of another class is known as inheritance. Inheritance is process of object reusability. E.g., A Child acquire property of Parents.
- **Polymorphism**: Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means "many forms". A subclass can define its own unique behavior and still share the same functionalities or behavior of its parent/base class. A subclass can have their own behavior and share some of its behavior from its parent class not the other way around. A parent class cannot have the behavior of its subclass. One function behaves different forms. In other words, "Many forms of a single object is called Polymorphism." E.g., Person behaves SON in house at the same time that person behaves EMPLOYEE in office.





## Pillars from Implementation Point of view

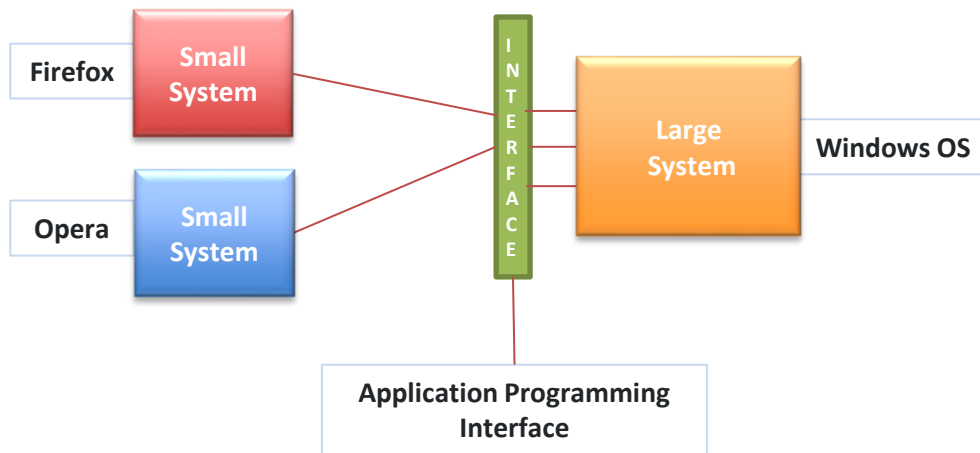
- **Encapsulation** means hiding data like using getter and setter etc. Encapsulation is the process of hiding information details and protecting data and behavior of an object from misuse by other objects. In Java, encapsulation is done using access modifiers (public, protected, private) with classes, interfaces, setters, getters.
- **Abstraction** means hiding implementation using abstract class and interfaces etc. (It is providing a generalization (say, over a set of behaviors). Abstraction is the process of modeling real things in the real world into programming language. In Java, the process of abstraction is done using interfaces, classes, abstract classes, fields, methods and variables. Everything is an abstraction.
- **Inheritance** means create new class from existing class.
- **Polymorphisms** means invoking any child class object method implementation (overridden or non overridden) using implementing interface, parent abstract class or parent concrete class object.

<https://www.codejava.net/java-core/the-java-language/what-is-encapsulation-in-java-the-what-why-and-how?fbclid=IwAR1i5TRxvnY74Ay7IP4InZVvkQQ9FWdIkWpeSV2gbhiX5rpABJzDGDj6xg>

<https://www.codejava.net/java-core/the-java-language/what-is-abstraction-in-java-the-why-and-the-truth>

## Pillars from Implementation Point of view cont..

- **Abstraction (or modularity) – Abstract Data Types (ADT) in Data Structure**
  - Types (data types) enable programmers to think at a higher level than the bit or byte, not bothering with low-level implementation.
  - For example, programmers can begin to think of a string as a set of character values instead of as a mere array of bytes.
  - Higher still, types enable programmers to think about and express interfaces between two of any-sized subsystems.
  - This enables more levels of localization so that the definitions required for interoperability of the subsystems remain consistent when those two subsystems communicate. [Source](#)
- Another good example of abstraction is the Java Virtual Machine (JVM), which provides a virtual or abstract computer for Java code to run on. It essentially takes away all of the platform specific components of a system and provides an abstract interface of "computer" without regard to any system.



## Pillars from Implementation Point of view cont..

- **Abstraction** : Abstraction means to show What part of functionality.
- **Encapsulation** : Encapsulation means to hide the How part of the functionality.

```
/// <summary>
/// We have an Employee class having two properties EmployeeName and EmployeeCode
/// </summary>
public class Employee
{
    public string EmployeeName { get; set; }
    public string EmployeeCode { get; set; }
    // Add new employee to DB is the main functionality, so are making it
    // public so that we can expose it to external environment
    // This is ABSTRACTION
    public void AddEmployee(Employee obj)
    {
        // "Creation of DB connection" and "To check if employee exists"
        // are internal details which we have hide from external environment
        // You can see that these methods are private, external environment just need "What" part only
        CreateDBConnection();
        CheckIfEmployeeExists();
    }

    // ENCAPSULATION using private keyword
    private bool CheckIfEmployeeExists()
    {
        // Here we can validate if the employee already exists
        return true;
    }

    // ENCAPSULATION using private keyword
    private void CreateDBConnection()
    {
        // Create DB connection code
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Employee obj = new Employee();
        obj.EmployeeName = "001";
        obj.EmployeeCode = "Raj";

        // We have exposed only what part of the functionality
        obj.AddEmployee(obj);
    }
}
```

## Pillars from Implementation Point of view cont..

- **Encapsulation:** Is hiding unwanted/un-expected/propriety implementation details from the actual users of object. e.g.

```
List<string> list = new List<string>();  
list.Sort(); /* Here, which sorting algorithm is used and hows its  
implemented is not useful to the user who wants to perform sort, that's  
why its hidden from the user of list. */
```

- **Abstraction:** Is a way of providing generalization and hence a common way to work with objects of vast diversity. e.g.

```
class Aeroplane : IFlyable, IFuelable, IMachine  
{ // Aeroplane's Design says:  
  // Aeroplane is a flying object  
  // Aeroplane can be fueled  
  // Aeroplane is a Machine  
}  
// But the code related to Pilot, or Driver of Aeroplane is not bothered  
// about Machine or Fuel. Hence,  
// pilot code:  
IFlyable flyingObj = new Aeroplane();  
flyingObj.Fly();  
// fighter Pilot related code  
IFlyable flyingObj2 = new FighterAeroplane();  
flyingObj2.Fly();  
// UFO related code  
IFlyable ufoObj = new UFO();  
ufoObj.Fly();  
// **All the 3 Above codes are generalized using IFlyable,  
// Interface Abstraction**  
// Fly related code knows how to fly, irrespective of the type of  
// flying object they are.  
  
// Similarly, Fuel related code:  
// Fueling an Aeroplane  
IFuelable fuelableObj = new Aeroplane();  
fuelableObj.FillFuel();  
// Fueling a Car  
IFuelable fuelableObj2 = new Car(); // class Car : IFuelable { }  
fuelableObj2.FillFuel();  
  
// ** Fueling code does not need know what kind of vehicle it is, so far  
// as it can Fill Fuel**
```

## Java Platform Standard Edition 8 Documentation (Oracle):

- <https://docs.oracle.com/javase/8/docs/>
- <https://docs.oracle.com/javase/tutorial/>
- <https://www.oracle.com/java/technologies/javase-jdk8-doc-downloads.html>

## Jdk 8 Features Page(Oracle):

- <https://www.oracle.com/java/technologies/javase/8-whats-new.html>

## OpenJdk 8 Documentation (OpenJdk):

- <https://devdocs.io/openjdk~8/java/lang/system>

## Jdk 8 Features Page(OpenJdk):

- <https://openjdk.org/projects/jdk8/features>

## Navigate OpenJdk Code Base:

The following notes are provided to help newcomers to javac navigate their way around the code base

- <https://openjdk.org/groups/compiler/doc/hhgtjavac/index.html#files>

## Java Keywords:

- <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/keywords.html>

## Literals:

- <https://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html>

## Install Jdk 8, 11, 17

Post Installation Commands:

**C:\Users\GD>where java**

```
C:\Program Files (x86)\Common Files\Oracle\Java\javapath\java.exe
C:\Program Files\Eclipse Adoptium\jdk-8.0.352.8-hotspot\bin\java.exe
C:\Program Files\Eclipse Adoptium\jdk-11.0.17.8-hotspot\bin\java.exe
C:\Program Files\Eclipse Adoptium\jdk-17.0.2.8-hotspot\bin\java.exe
```

**C:\Users\GD>where javac**

```
C:\Program Files\Eclipse Adoptium\jdk-8.0.352.8-hotspot\bin\javac.exe
C:\Program Files\Eclipse Adoptium\jdk-11.0.17.8-hotspot\bin\javac.exe
C:\Program Files\Eclipse Adoptium\jdk-17.0.2.8-hotspot\bin\javac.exe
```

**C:\Users\GD>java -version**

```
java version "1.8.0_351"
Java(TM) SE Runtime Environment (build 1.8.0_351-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.351-b10, mixed mode)
```

**C:\Users\GD>javac -version**

```
javac 1.8.0_352
```

**C:\Users\GangadharParde>java -fullversion**

```
java full version "1.8.0_351-b10"
```

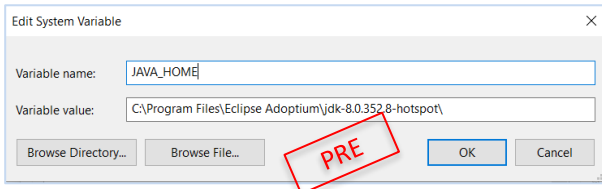
<https://www.oracle.com/java/technologies/javase/jdk8-naming.html>

Ensure to set JAVA\_HOME path inside your Windows env variables to right version of Jdk

## Issue: java -version and javac -version showing different versions

You might see issues of different version due to multiple version installed on your system.

Ensure you have one version of jdk configured under Path variable unlike below before "%SystemRoot%\system32"



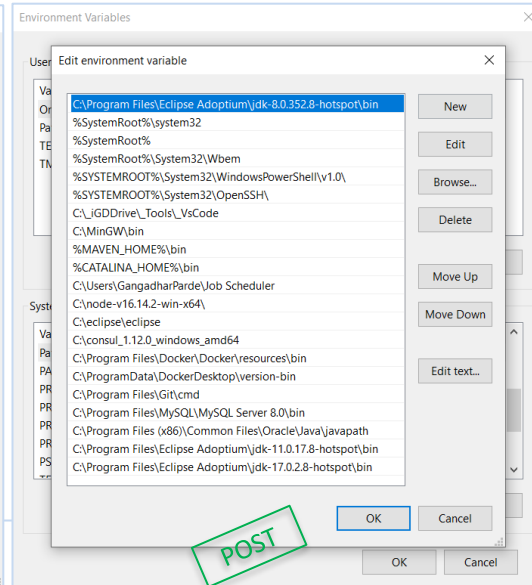
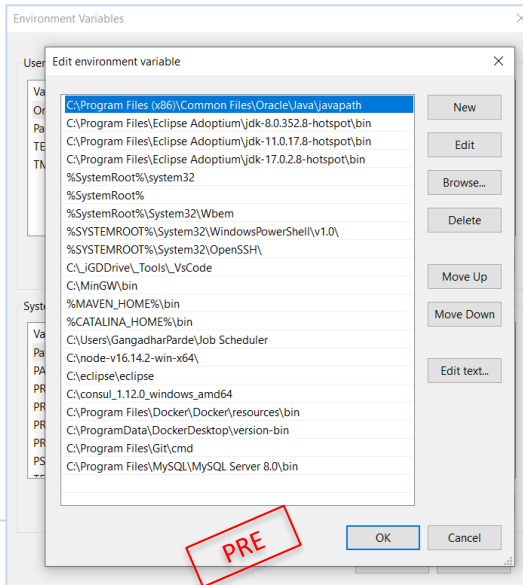
### Post Changes:

```
C:\Users\GD>javac -version
javac 1.8.0_352
```

```
C:\Users\GD>java -version
openjdk version "1.8.0_352"
OpenJDK Runtime Environment
(Temurin)(build 1.8.0_352-b08)
OpenJDK 64-Bit Server VM (Temurin)(build
25.352-b08, mixed mode)
```

### Steps:

1. Go to Windows search bar > type > env
2. click menu "Edit the system environment variables"
3. Advanced tab > click on button "Environment Variables..."
4. Under "System Variable" > select "Path" > click "Edit"
5. Looking for Java path such as "C:\Program Files\Java\jdk-19\bin"
6. Move the path using "Move Up" or "Move Down" button to the position before "%SystemRoot%\system32" Path
7. OK > OK



# Data Types

Data types are divided into two groups:

**Primitive data types** - includes byte, short, int, long, float, double, boolean and char

**Non-primitive data types** - such as String, Arrays and Classes (you will learn more about these in a later chapter)

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false



1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator ( & Arithmetic Assignment Operators)
4. Relational Operators
5. Logical Operators
6. Ternary Operator
7. Bitwise Operators
8. Shift Operators
9. instanceof operator

<https://processing.org/reference/leftshift.html#:~:text=Each%20shift%20to%20the%20left,together%20into%20one%20larger%20number.>

<https://processing.org/reference/rightshift.html>

<https://www.jdoodle.com/online-java-compiler/>

## **Unary:**

*Operator Operand e.g.  $-(a)$  where  $a=20$*

*Operand Operator e.g.  $a++$  or  $a--$ ;*

## **Binary:**

*Operand Operator Operand e.g.  $a+b$*

## **Ternary:**

*Operand ? Operand : Operand*

*Condition ? Value : Value*

# Literals, Variable, Variable Scopes

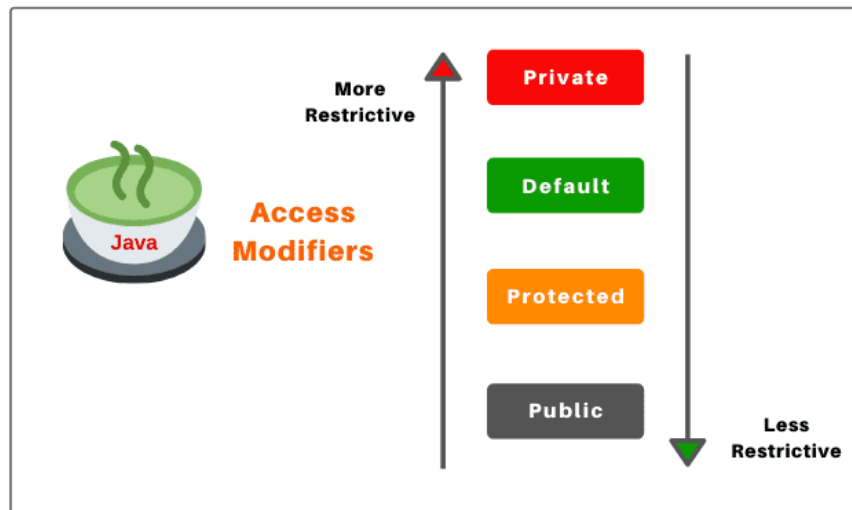
---

There are 4 types of variables in Java programming language:

1. Instance Variables (Non-Static Fields)
2. Class Variables (Static Fields)
3. Local Variables
4. Parameters

## Access Modifiers

	PRIVATE	DEFAULT	PROTECTED	PUBLIC
Same class	Yes	Yes	Yes	Yes
Same package Subclass	No	Yes	Yes	Yes
Same package Non-subclass	No	Yes	Yes	Yes
Different package Subclass	No	No	Yes	Yes
Different package Non-subclass	No	No	No	Yes



## Appendix

<https://stackoverflow.com/questions/742341/difference-between-abstraction-and-encapsulation>

<http://pankajtiwarii.blogspot.com/p/oops-abstraction-encapsulation.html>

<http://www.tonymarston.co.uk/php-mysql/abstraction.txt>

***Duke is the Java mascot***

<https://openjdk.org/projects/duke/>

<https://hg.openjdk.java.net/duke/duke/>

