What is Java?

* Java is an [object-oriented](https://www.w3schools.in/java-tutorial/object-oriented-programming-oops/) programming language developed by Sun Microsystems, and it was released in 1995.
* James Gosling initially developed Java in Sun Microsystems (which was later merged with Oracle Corporation).
* Java is a set of features of C and C++. It has obtained its format from C, and OOP features from C++.
* Java programs are platform independent which means they can be run on any operating system with any processor as long as the [Java interpreter](https://www.w3schools.in/java-tutorial/java-virtual-machine/) is available on that system.
* Java code that runs on one platform does not need to be recompiled to run on another platform; it's called write once, run anywhere(WORA).
* [Java Virtual Machine (JVM)](https://www.w3schools.in/java-tutorial/java-virtual-machine/) executes Java code, but it has been written in platform-specific languages such as [C](https://www.w3schools.in/c/intro/)/[C++](https://www.w3schools.in/cplusplus/intro/)/ASM, etc. JVM is not written in Java and hence cannot be platform independent, and Java interpreter is a part of JVM.

Where is Java being used?

* **JSP -**In Java, JSP (Java Server Pages) is used to create dynamic web pages, such as in PHP and ASP.
* **Applets -**Applets are another type of Java programs that are implemented on Internet browsers and are always run as part of a web document
* **J2EE -**Java 2 Enterprise Edition is a platform-independent environment that is a set of different protocols and APIs and is used by various organizations to transfer data between each other.
* **JavaBeans -**This is a set of reusable software components that can be easily used to create new and advanced applications.
* **Mobile** **-** In addition to the above technology, Java is widely used in mobile devices nowadays, many types of games and applications are being made in Java.

Types of Java Application

* **Web Application -** Java is used to create server-side web applications. Currently, Servlet, JSP, Struts, JSF, etc. technologies are used.
* **Standalone Application -** It is also known as the desktop application or window-based application. An application that we need to install on every machine or server such as media player, antivirus, etc. AWT and Swing are used in java for creating standalone applications.
* **Enterprise Application -** An application that is distributed in nature, such as banking applications, etc. It has the advantage of high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.
* **Mobile Application**

Features of Java

* **Object-Oriented**
* **Platform independent**
* **Secure -**Java provides a wide range of protection from viruses and malicious programs.  It ensures that there will be no damage and no security will be broken.
* **Portable -**Java provides us with the concept of portability. Running the same program with Java on different platforms is possible.
* **Robust -**During the development of the program, it helps us to find possible mistakes as soon as possible.
* **Multi-threaded -**The multithreading programming feature in Java allows you to write a program that performs several different tasks simultaneously.
* **Distributed -**Java is designed for distributed Internet environments as it manages the TCP/IP protocol.

**Classes and Objects**

public class MyClass {

public static void main(String[] args) {

System.out.println("Hello World");

}}

# What is static keyword ?

Static is a keyword that acts as a non-access modifier in Java that is used mainly to manage memory. The variable or Method that are marked static belongs to the Class rather than to any particular instance. A Static method cannot access an instance variable. If a Class contains any static blocks, then that block will be executed only when the Class is loaded in JVM.

# What is final ?

Once a variable is assigned with the keyword final, it always contains the same exact value.

# What is super in Java ?

Super is a keyword of Java which refers to the immediate parent of a class and is used inside the subclass method definition for calling a method defined in the superclass.

super.<method-name>();

# Types of Inner Class

* Local - Method Inner class
* Anonymous Inner class

public class classA {

void fun() {

int val = 84;

// local-method inner class

class InnerMeth {

public void disp() {

System.out.println("Inner class method: "+val);

}

} // end of inner class

// Accessing the inner class

InnerMeth in = new InnerMeth(); in .disp();

}

public static void main(String argu[]) {

classA out = new classA();

out.fun();

System.out.println("Program done….");

}}

abstract class anonInner {

public abstract void fun();

}

public class outerA {

public static void main(String argu[]) {

anonInner in = new anonInner() {

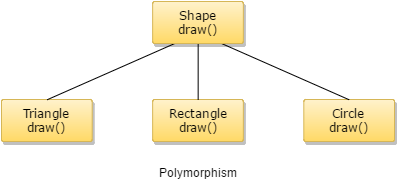
public void fun() {

System.out.println("Anonymous Inner class executed….");

}};

in .fun();}}

Polymorphism





It has two types…………………………………………………………………………………….  
1)StaticPolymorphism  
2) Dynamic Polymorphism

* Method Overloading(Compile time Polymorphism)
* Method Overriding(Run time Polymorphism)

class parent {

public void work() {

System.out.println("Parent is under retirement from work.");

}

}

class child extends parent {

public void work() {

System.out.println("Child has a job");

System.out.println(" He is doing it well");

}

public static void main(String argu[]) {

child c1 = new child();

c1.work();

}

}

Abstraction

* Likewise in Object-oriented programming, abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user.
* Java provides the concept of abstraction through Abstract classes and interfaces. A class containing the keyword abstract in its declaration creates the abstract class. It may or may not contain any abstract methods within it.
* When a class is classified as abstract, it cannot be instantiated. For using a class as abstract, it needs to be inherited from another class which has the abstract method implementation. It is to be noted that if a class has a minimum of one method as abstract, then the class has to be declared as abstract.

abstract class cycle { abstract void work(); }

class HeroCycle extends cycle {

void work(){ System.out.println("Selling good");}

public static void main(String argu[]) {

cycle o = new HeroCycle();

o.work();

System.out.println("Code executed");

}}

Encapsulation

In simple terms, encapsulation is a way of packaging data and methods together into one unit. Encapsulation gives us the ability to make variables of a class keep hidden from all other classes of that program or namespace.

* Class fields can be made read-only or write-only
* Programmers can have full control over what data storage and manipulation within the class

Inheritance

It is the mechanism of acquiring all the properties and behavior of one class to another, i.e., acquiring the properties and behavior of child class from the parent class. This concept was built to achieve the advantage of creating a new class that gets built upon an already existing class(es). It is mainly used for code **reusability** within a Java program.

class base

{

.....

.....

}

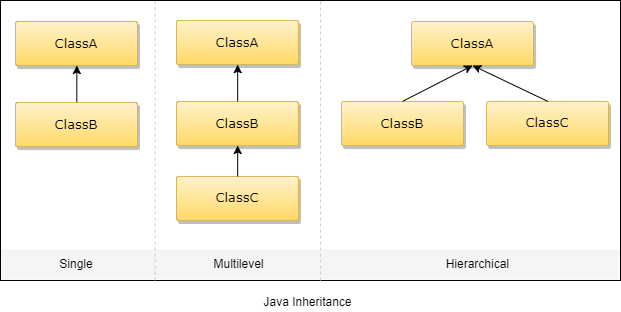
class derive extends base

{

.....

.....

}



## Single Inheritance

class Teacher {

void teach() {

System.out.println("Teaching subjects");

}

}

class Students extends Teacher {

void listen() {

System.out.println("Listening to teacher");

}

}

class CheckForInheritance {

public static void main(String args[]) {

Students s1 = new Students();

s1.teach();

s1.listen();

}

}

## Multi-Level Inheritance

class Teacher {

void teach() {

System.out.println("Teaching subject");

}

}

class Student extends Teacher {

void listen() {

System.out.println("Listening");

}

}

class homeTution extends Student {

void explains() {

System.out.println("Does homework");

}

}

class CheckForInheritance {

public static void main(String argu[]) {

homeTution h = new himeTution();

h.explains();

d.teach();

d.listen();

}

}

## Hierarchical Inheritance

class Teacher {

void teach() {

System.out.println("Teaching subject");

}

}

class Student extends Teacher {

void listen() {

System.out.println("Listening");

}

}

class Principal extends Teacher {

void evaluate() {

System.out.println("Evaluating");

}

}

class CheckForInheritance {

public static void main(String argu[]) {

Principal p = new Principal();

p.evaluate();

p.teach();

// p.listen(); will produce an error

}

}

Interface

class X extends Y extends Z

{

. . . . . .

}

That is Java classes cannot have more than one superclass.

public interface IntrfaceName

{

//Any number of final, static fields

//Any number of abstract method declarations

}

access interface\_name{

return-type method1(parameter list);

return-type method2(parameter list);

type final-varName1=value;

//         . . . . . . . .

}

## Example of Interface

import java.util.\*;  
import java.lang.\*;  
import java.io.\*;

interface shapeX {

public String base = "shape1";

public void Draw();

}

interface shapeY extends shapeX {

public String base = "shape2";

public void Draw2();

}

class shapeG implements shapeY {

public String base = "shape3";

public void Draw() {

System.out.println("Drawing Circle here:" + base);

}

@Override

public void Draw2() {

System.out.println("Drawing Circle here:" + base);

}

}

public class Main {

public static void main(String[] args) {

shapeG circleshape = new shapeG();

circleshape.Draw();

circleshape.Draw();

}

}

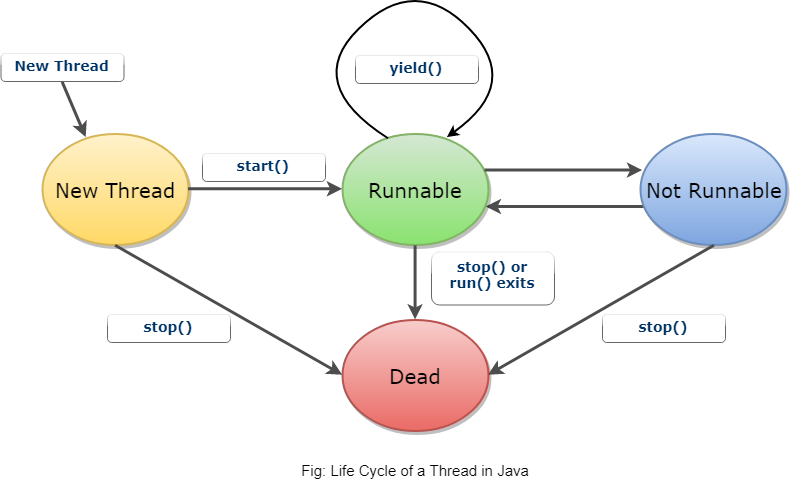
Multithreaded

Multithreaded programming contains two or more parts that can run **concurrently**. Each piece of such a program is called a thread, and each thread defines a separate path of execution. Thus multithreading can be said as a particular version of multitasking.

## What is ****multitasking?****

* **Process-based multitasking-** It is also called multiprocessing where each process has its address in memory, i.e., each process allocates separate memory area.
* **Thread-based multitasking-** This thread-based multitasking is also termed as multithreading where threads share the same address space.

## Life Cycle of Thread



class MultiThread extends Thread{

public void run(){

System.out.println("Running Thread Name: "+ this.currentThread().getName());

System.out.println("Running Thread Priority: "+ this.currentThread().getPriority());

}

}

public class MultiThrd {

public static void main(String[] args) {

MultiThread multiThread1 = new MultiThread();

multiThread1.setName("First Thread");

multiThread1.setPriority(Thread.MIN\_PRIORITY);

MultiThread multiThread2 = new MultiThread();

multiThread2.setName("Second Thread");

multiThread2.setPriority(Thread.MAX\_PRIORITY);

MultiThread multiThread3 = new MultiThread();

multiThread3.setName("Third Thread");

multiThread1.start();

multiThread2.start();

multiThread3.start();

}}

Overriding

class college {

public void move() {

System.out.println("College is open");

}

}

class univ extends college {

public void move() {

System.out.println("University is open too");

}

}

public class stud {

public static void main(String args[]) {

college a = new college();

college b = new univ();

a.move();

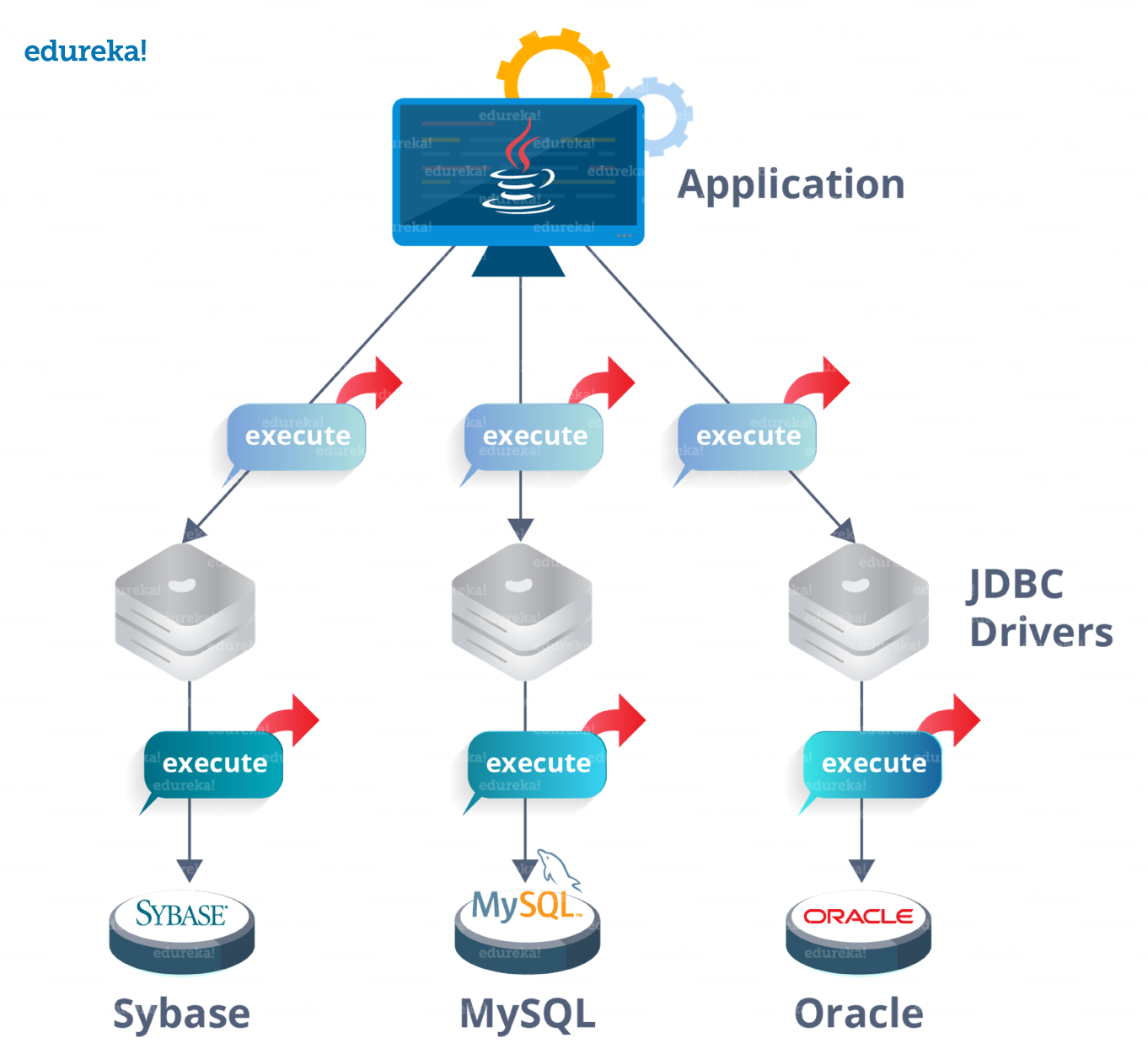
b.move();

}

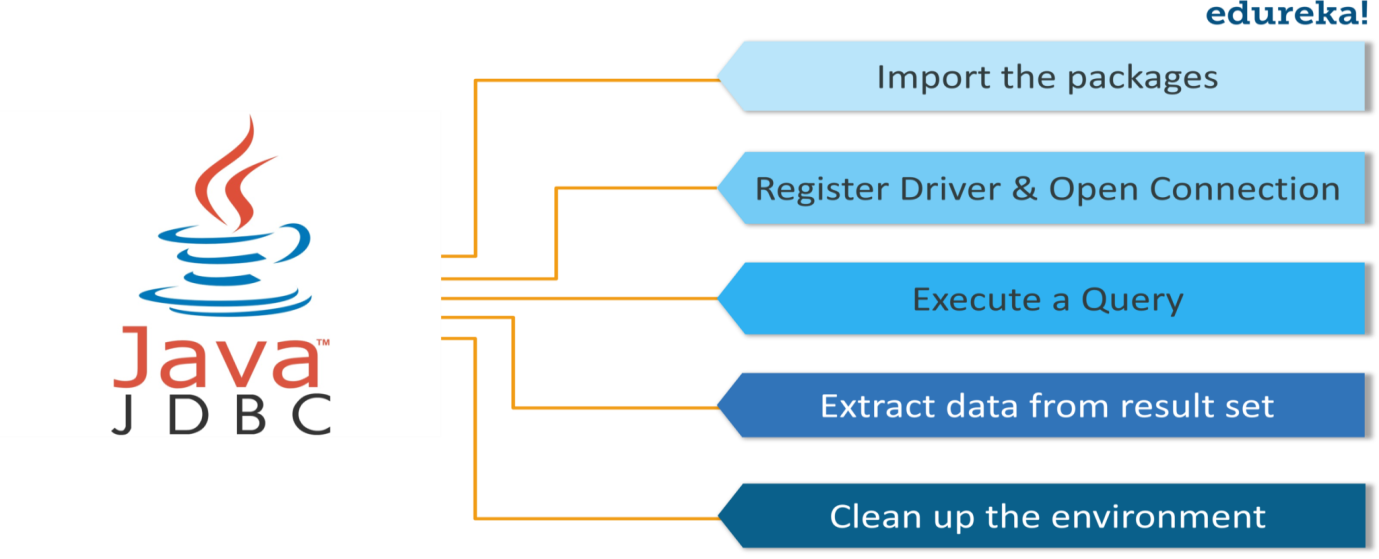
}

**Advanced Java**

### JDBC



* ***DriverManager***is used to manage a list of database drivers. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
* ***Driver*** is an interface that handles the communications with the database server. It also abstracts the details associated with working with Driver objects.
* ***Connection***is an interface that consists all the methods required to connect to a database. The connection object represents communication context, i.e., all communication with the database is through connection object only.



1. **Import the packages:**You need to include the packages containing the JDBC classes needed for database programming. Most often, using *import java.sql.\** will suffice.
2. **Register the JDBC driver:** Here you have to initialize a driver so that you can open a communication channel with the database.
3. **Open a connection:** Here, you can use the *getConnection()* method to create a Connection object, which represents a physical connection with the database.
4. **Execute a query:**Requires using an object of type Statement for building and submitting an SQL statement to the database.
5. **Extract data from result set:** Requires that you use the appropriate *getXXX()* method to retrieve the data from the result set.
6. **Clean up the environment:** Requires explicitly closing all database resources versus relying on the JVM’s garbage collection.

Java Mysql link  
<https://jar-download.com/artifacts/mysql/mysql-connector-java/8.0.11/source-code>  
<https://github.com/kuntal-python/CoreJava/blob/master/DBMSJAVA/src/org/DBAccess.java>  
<https://drive.google.com/drive/folders/11CmZQ7UHlmSMlj8v1NaNQg1mhJxjiKUC>