What is Java?

* It is a programming language.
* It is high level, robust, object oriented and secure programming language.
* It is platform independent. It was introduced in 1995.

<https://docs.oracle.com/javase/8/docs/api/index.html>

OpenJDK

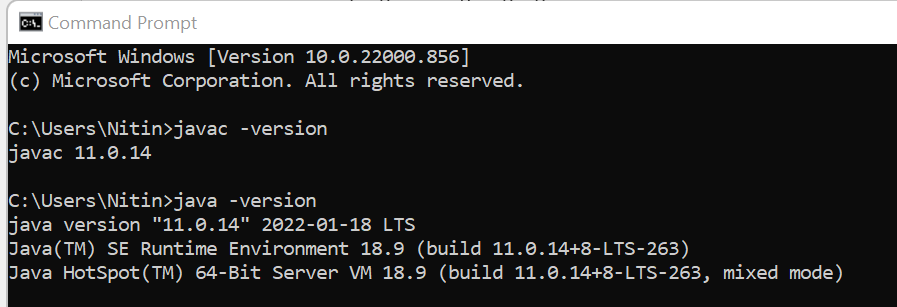
<https://www.oracle.com/java/technologies/downloads/>

Oracle JDK

<https://www.oracle.com/java/technologies/downloads/>

after installation - verify the installation

go to command prompt / terminal and ensure the version for javac & java are the same.



If it doesn’t work,

Set the environment variable that is PATH.

set PATH = %PATH%; C:\Program Files\Java\jdk-11.0.14\bin

alternatively we can go to Edit System Variable and add Java Home path to Path variable.

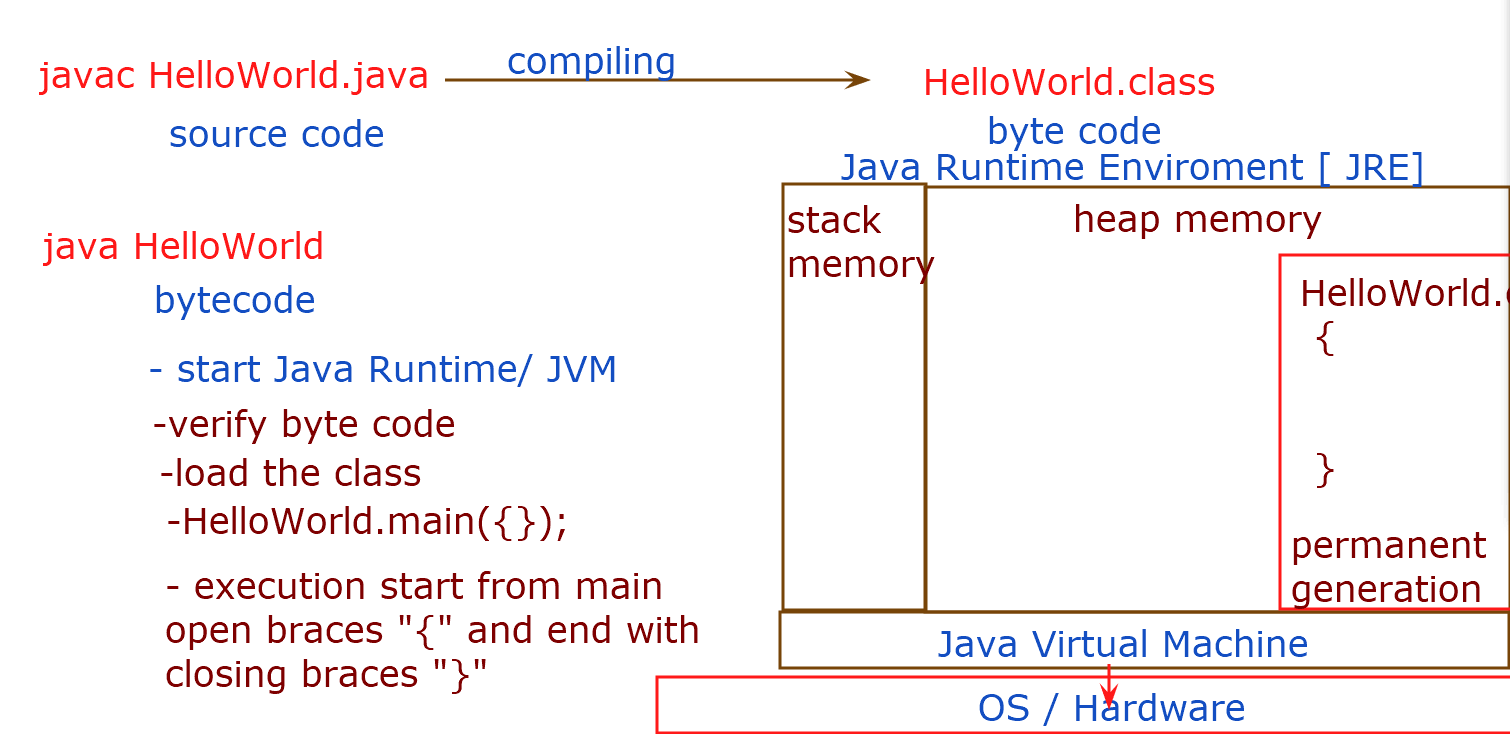
MacOS [ pls check the java home path ]

export PATH = $PATH: /usr/java/jdk-11.0.14/bin

javac === Java compiler / it is used to translate high-level Language to Byte Code.

java == Java Runtime / it is responsible for starting JVM and executing Java Main class that is the starting point of your java application.

**Write Once and Run Anywhere**



**Types of Java Edition**

1. Java Standard Edition [ Java SE ]
   * Desktop applications
   * Console Application [ No GUI ]
   * Java Libraries
2. Java Enterprises Edition [ Java EE ]
   * Web Application
   * Distributed Applications [ Enterprise Application]
3. Java Micro Edition [ Java ME ]
   * Mobile Application
   * Java TV
   * Java Card [ program information for smart card chipset]

**Object Oriented Programming**

1. Encapsulation & Data Hiding
2. Inheritance
3. Abstraction
4. Polymorphism
5. **Encapsulation & Data Hiding**

Wrapping up data & behavior into one capsule / block is known as encapsulation. A class is a way to encapsulate the state[ data ] & behavior.

This is to form blue print.

Data hiding - To hide the data from direct accessibility to prevent direct modification.

Graphical user interface, text

Description automatically generated

**Syntax for Creating Object**

<ClassName> ref-variable = new <ClassName>();

* Java is case Sensitive
* Java follows certain naming convention

1. ClassName - upper camel case
2. Field members [ state ] & methods [ behavior ] – lower camel case
3. Constant variable -- UPPER CASE

Text

Description automatically generated

**Types of Variables**

1. Primitive Type [ 8 types ]
   1. byte 8 bits -128 to 127
   2. short 16 bits -32768 to 32767
   3. **int**  32 bits -2147483648 to -2147483647
   4. long 64 bits
   5. float 32 bits
   6. **double**  64 bits
   7. char 16 bits Unicode char ‘A’
   8. boolean \*1 bit [ actual size may vary depending on OS] true/false

Any number with a precision literal value like 1.0 , is by default double type

All whole numbers 10 are by default-int

1. Reference Type or Non-Primitive

Class, Interface, Enum & Array types of variables are known as reference types.

Diagram, schematic

Description automatically generated

**Scope of Variables**

1. Instance Variable / Field members – Accessible in all methods in the given class. All the instance variables get initialized with default values at the time of instantiation.

byte, short, int , long ------------------------------ 0 [ default value ]

float, double ------------------------------------------ 0.0

boolean ----------------------------------------------- false

char --------------------------------------------------- ‘\u’

all reference type , the default value is null.

1. Local Variables - Accessible only in the block that they are declared in. you must initialize it before you use it.
2. Static Variables / Class Variable – Accessible by all instances and maintains only copy per JVM.

**Static Variables** – they get memory allocated at the time of class loading. They belong to class memory. It is initialized with a default value.

There is the only copy of a static variable is created per JVM. This can be shared by all instances of a class.

**Static Methods:**

You can use a static keyword to declare a method. This method can access only static variables. You will have only one copy of static context per JVM.

Relational Operators

>

<

<=

>=

==

!=

Arithmetic Operators

+

-

\*

/

%

++

--

Logical Operator

&&

||

!

Access Modifier “private” keyword can be used to hide.

**Constructor** is a method with the same name as classname and no return type. It is invoked automatically at the time of instantiation. It is used to initialize the fields.

A). No Argument Constructor [ default constructor ] is supplied by the compiler to the byte code of a class only if no constructors were added by the developer.

B) No default constructor will be added if you have your own constructor. You may need to add it manually in your code.

**Inheritance**

**public** **class** Manager **extends** Employee {

**private** String deptName;

**public** Manager(**int** id, String name, **double** salary, String deptName) {

**super**(id,name,salary);

**this**.deptName=deptName;

}

**public** String getDeptName() {

**return** deptName;

}

**public** **void** setDeptName(String deptName) {

**this**.deptName = deptName;

}

}

**Method Overriding**

Re-writing the super class method that is inherited, is known as method overriding.

* Method in subclass must have the same name, signature [ arguments types, numbers and sequence] and return type.
* Overridden method can not have less accessibility than super class method.

**Method Overloading**

Writing multiple methods with the same name but with different/same functionality in a class. Constructors can be overloaded as well.

Rule 1 – there must be difference in method’s signature.

TNS / TNO = Type, Number and Sequence/Order of Arguments.

System.out.println();

String s=”Nitin”;

System.out.println(s);

int num=10;

System.out.println(num);

Employee e1=new Employee(1,”Alex”,1290.00);

System.out.println(e1);

class Maths{

public static int sum(int x, int y){

return x+y;

}

public static int sum(int x, int y, int z){

return x+y;

}

public static double sum(double x, double y){

return x+y;

}

public static double sum(double x, int y){

return x+y;

}

public static double sum(int y, double x){

return x+y;

}

}}

**Conditional Statements**

if(condition/s){

//statements when the condition is true

}

if(condition/s){

//statements when the condition is true

}else{

//statements when the condition is false

}

if(condition1){

//statements when the condition1 is true

}else if(condition2){

//statements when the condition2 is true

}else if(condition\_N){

//statements when the conditionN is true

}else{

//statements when the none of the above are true

}

\*\*\*\*\*\* Nesting of If is possible as n when required.

switch(variable){

case value1:

//statement when variable==value1

If(var==value1){

//statement when variable==value1

}else if(var==value2){

//statement when variable==value2

}else if(var==value\_N){

}else{

break;

case value2:

//statement when variable==value2

break;

case value\_N:

//statement when variable==valueN

break;

default:

//statement when none were satisfied

}

\*\*\*\*values in the switch are either literals or constant variables.

If(var==value1 or var=value2){

//statement when either conditions are true

}

switch(variable){

case value1:

case value2:

//statement when variable==value1 or value2

break;

case value\_N:

//statement when variable==valueN

break;

default:

//statement when none were satisfied

}

**Loops :**

**repeating the execution of statements as long as conditions are true.**

while(condition/s){ // iterate the statements zero or more times.

//statements

}

do{ // iterate the statements one or more times.

//statements

}while(condition/s);

for(initialize var ; conditions ; increment/decrement){

statements

}

**int** i=1;

**while**(i<=10) {

System.***out***.println(i);

i++;

}

**int** i=1;

**do** {

System.***out***.println(i);

i++;

}**while**(i<=10);

**for**(**int** i=1; i<=10;i++) {

System.***out***.println(i);

}

**Array**

Array is a set of a finite number of elements of same type. Each elements share the same name as array.

**Declaring an Array**

Type[] arrayName;

**Instantiating An Array**

arrayName=new type[no. of elements];

**Declaration & Instantiation**

type[] arrayName=new type[no. of elements];

**Declaration ,Instantiation & Inialization**

type[] arrayName={value1, value2, value3, ValueN};

\*\*\* number of values will determine the number of elements.

**\*\*\* elements gets initialized with the default values as per type.**

**Example**

int[] numbers=new int[5];

numbers[0]=10;

numbers[1]=20;

numbers[4]=50;

**OR**

int[] numbers={1,2,3,4,5};

Diagram

Description automatically generated

for(int i=0; i<numbers.length ; i++){

System.out.println(numbers[i]);

}

OR

Enhanced For loop / for Each loop

for(int x:numbers){ // introduced first time in Java 5

System.out.println(x);

}

**Multi-Dimensional Array**

type[][] arrayname=new type[no. of array][no. of elements];

type[][] arrayname=new type[row][col];

int [][] numbers =new int[4][4];

Table, calendar

Description automatically generated with medium confidence

Employee[] empList=**new** Employee[4];

empList[0]=**new** Employee(1,"Nitin",2000.00);

empList[1]=**new** Employee(2,"Alex",3000.00);

empList[2]=**new** Employee(3,"Dereje",4000.00);

empList[3]=**new** Employee(4,"Gerges",5000.00);

// for(Employee e:empList) {

// e.print();

// System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

//

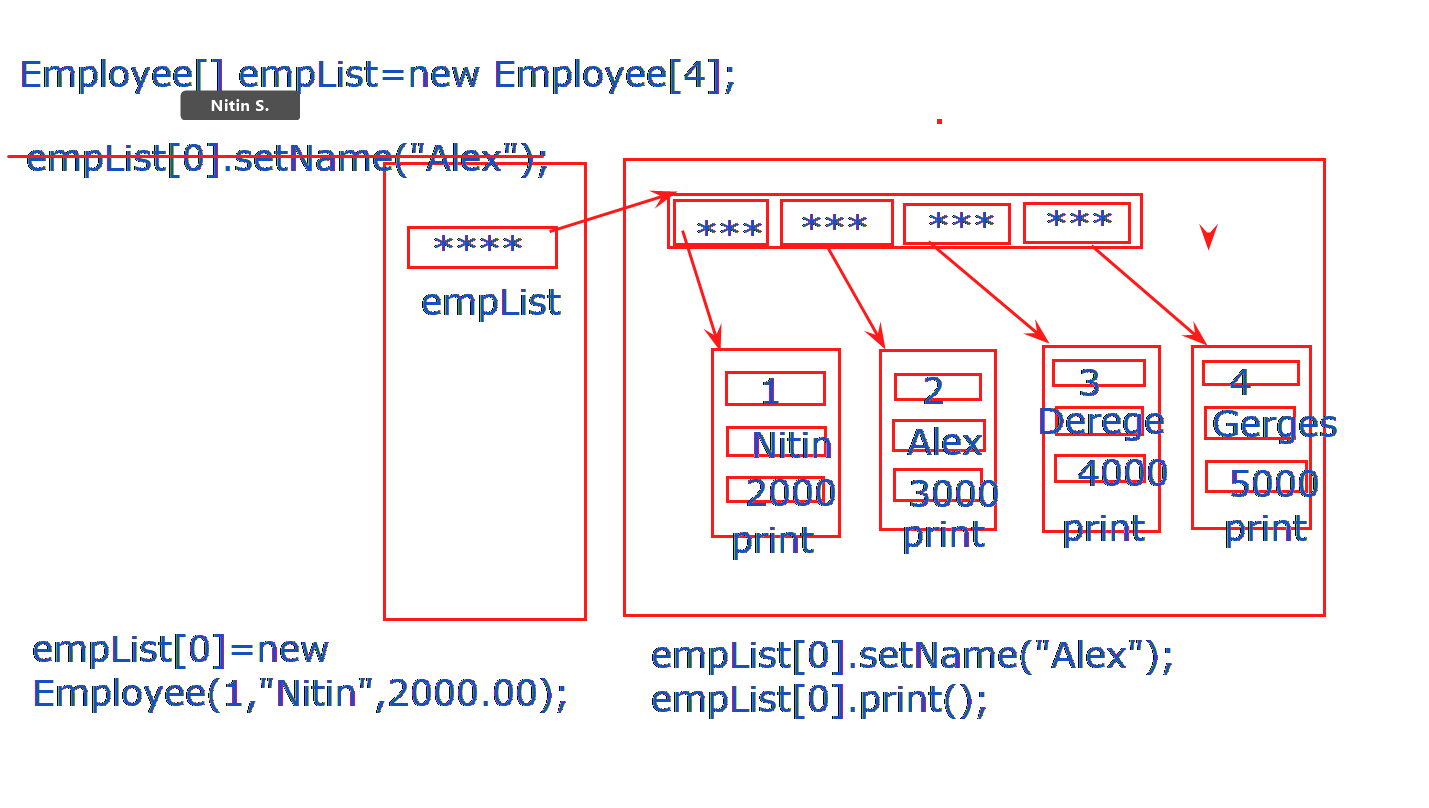
// }

**for**(**int** i=0; i<4; i++) {

empList[i].print();

System.***out***.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}



**Abstraction & Polymorphism**

* **Abstract Class :** A class that need not be instantiated and can be extended only, is an abstract class.

**public** **abstract** **class** Employee {

**private** **final** **int** id;

**private** String name;

**private** **double** salary;

**public** Employee(**int** id, String name, **double** salary) {

**this**.id=id;

**this**.name=name;

**this**.salary=salary;

}

//setters n getters

}

* **Abstract Method.** A method that must be overridden by all sub-classes and can be declared without definition[body] in an abstract class, is an abstract method.
  + **Subclass has only two choice :**
    - **To override/implement the abstract method**
    - **Or declare the class as the abstract**

---- see the examples – WalkingApp

**Polymorphism**

Poly = many

Morphism = faces

Employee e= ?;

e.print(); //print() must exist in Employee class

at the run time the code that will execute it can be either from Manager Class [ if overridden ] or from Employee class.

\*\*\*\* Virtual Method Invocation / Dynamic method call.

private – accessible with in the class.

default - accessible with-in the class and also by classes in the same package.

protected – accessible with-in the class, class in the same package and by subclasses from different packages.

public - accessible by all.

**Interfaces :**

* Contains only abstract methods. This is by default.
* All the methods in interfaces are by default public.
* A variable is by default public final static.

public interface A{

int x=10;

public void m1();

void m2();

}

Diagram

Description automatically generated

interface Returnable{

public void doReturn();

}

|  |  |
| --- | --- |
| class Shirt extends Clothing implements Returnable{  ---- properties----  public vod doReturn(){  S.o.p(“5 days return policy”);  }  } | class Trouser extends Clothing implements Returnable{  ---- properties----  public vod doReturn(){  S.o.p(“7 days return policy”);  }  } |
| class Bellies extends Footwear implements Returnable{  ---- properties----  public vod doReturn(){  S.o.p(“10 days return policy”);  }  } | class Flats extends Footwear implements Returnable{  ---- properties----  public vod doReturn(){  S.o.p(“15 days return policy”);  }  } |

Returnable r=?

new Shirt();

new Trouser();

new Flats();

new Bellies();

Clothing c= new Shirt();

if(c instanceof Shirt) {

Returnable r=(Returnable) c;

}

Returnable r=(Returnable) c; // ClassCastException will be throawn by Java Runtime if c refers to the object not implementing Returnable.