

**What is java?**

It is a High Level, Object Oriented, Strictly Typed, Platform Independent, General Purpose Programming Language Developed by Mr James Gosling and his team at Sun Microsystems (now Taken over by Oracle) in 1995-96. It’s also a Platform and Technology.

Basic Terms You Need to know before starting:

Programming Language:- It Provides Syntax and Semantics to create computer program. OR A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks

**Types of Programming Languages: -**

* Low-Level e.g., (Machine and Assembly Language)
* Mid-Level e.g., (C Language)
* High-Level e.g., (COBOL, FORTAN, PASCAL, B, CPL, C++, JAVA, C# etc...)

Low level programming language are those language whose program are directly interact with hardware. They can be easily understand by machine but cannot be easily understand by human.

Program: A computer program may be defined as the set of instructions in static state OR we can say that it is a chain of steps OR a set of activities which combine together to perform a specific task.

Process: It’s the Active state of a program OR we can say that program in execution is called a process.

Thread: It’s a Sub-Process. Process Inside process or light weight process

Translator: A translator is a program that converts a computer program from one language to another (in general from High Level Language program to Machine Language Program). It takes a program written in source code and converts it into machine code.

There are three type of Translator:

1. Compiler
2. Assembler
3. Interpreter

Java is the compile as we as interpret programming language. Where as c and c++ is the only compile programming language.

Software:- Software is a set of instructions, data or programs used to operate computers and execute specific tasks. Opposite of hardware, which describes the physical aspects of a computer, software is a generic term used to refer to applications, scripts and programs that run on a device.

From software development point of **a software is the combination of requirement specification (SRS) + Implementation + Documentation**

Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality Software. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

A typical Software Development Life Cycle has the following stages:

[1. Planning](https://arkbauer.com/blog/software-development-life-cycle-sdlc/#1planning)

[2. Requirement analysis](https://arkbauer.com/blog/software-development-life-cycle-sdlc/#2-requirement-analysis)

[3. Design](https://arkbauer.com/blog/software-development-life-cycle-sdlc/#3-design)

[4. Implementation](https://arkbauer.com/blog/software-development-life-cycle-sdlc/#4-implementation)

[5. Testing and Integration](https://arkbauer.com/blog/software-development-life-cycle-sdlc/#5-testing-and-integration)

[6. Maintenance](https://arkbauer.com/blog/software-development-life-cycle-sdlc/#6-maintenance)

### Graphical representation of the various stages of a typical SDLC.

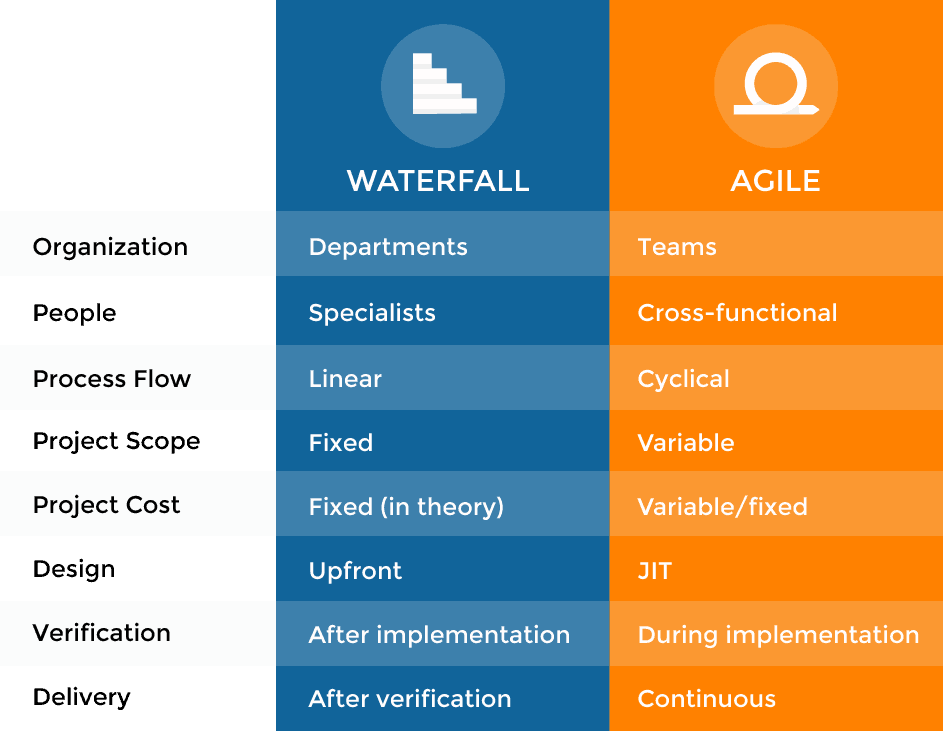
### Software development life-cycle (SDLC) | Arkbauer

Sdlc models: There are quite a few software development life-cycle models out there that you can follow and use. Each model follows a series of unique steps, with the ultimate aim of ensuring the success of the software development.

**Here are some of the most popular SDLC models in industry:**

* Waterfall also known as Traditional model
* Agile
* Iterative
* Spiral
* Rapid Application Development
* Prototyping

Waterfall v/s agile (Quick Differences)

****

Platform:

It is the environment on which our program is executed. It could be a software, hardware of a combination of both.

1. Software :- OS ,Server
2. Hardware :- CPU ,Microcontroller
3. Software + Hardware (Embedded Systems)

\*\*A Programming language is called platform independent if its program compiled on one platform can be executed on other platforms without changing their object code.in short it’s abbreviated as **CORA: Compile Once Run Anywhere**

**Programming methodology or programming paradigm**

A programming methodology deals with providing a way to consider and manage the development, design, implementation, and testing of a piece of software.

**OR**

A programming methodology is just a programming practice to help you plan and structure your code in a more defined way.

**Types of Programming Methodology:-**

1. Procedural Oriented (Top to Bottom Approach)
2. Object Oriented, Object Based, Pure Object Oriented. (Bottom to Top Approach)
3. Aspect Oriented (Bottom to Top Approach)

Example of POPL: C Language

Example of OOPL: Java, C++, Simula etc.

Example of AOPL: Spring.

Problems with C language

C is a procedural programming language and C++ is Object Oriented Programming Language.

1. No data hiding
2. It doesn’t support the concept of real world
3. No modularity
4. Complexity
5. Tight coupling
6. Platform Dependent

Advantages of C++ (OOPs) over C

1. Data Hiding 2. Modularity

3. Reusability 4. Simple

5. Loose Coupling 6. Because of OOPs it supports the concepts of real world

Pillars of OOPs: - Polymorphism Inheritance Encapsulation

Difference between java and c++

Java is platform independent while C++ is platform dependent.

Java has automatic garbage collection whereas C++ have destructor

Java does not support the explicit use of pointers whereas C++ have pointers

Java is very much used on internet while C++ is not suitable for internet programming

Java is compiled as well as interpreted programming language while C++ is only compiled.

Platform:-

It is the place on which our program is going to be executed it could be a hardware or software or combination of both.

**Type of Platform:-**

1. Software- O.S
2. Hardware- Micro controller or micro processor
3. Software + Hardware

What is Platform Independency: -

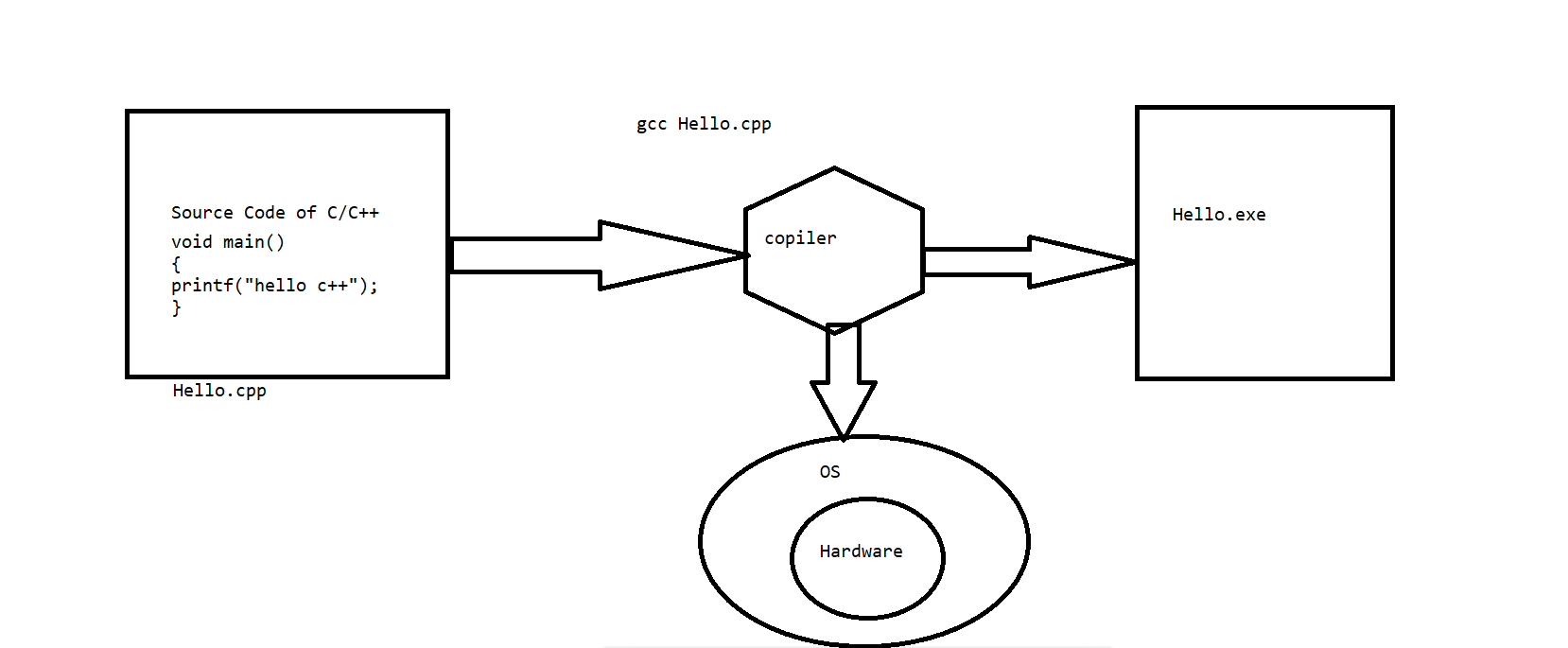
Any programming language is called platform independent if Its program is compiling on one platform and same program can be executed on another platform without changing It’s object code. In other word we can say that CORA (compile once run anywhere)

How C++ is Platform Dependent:

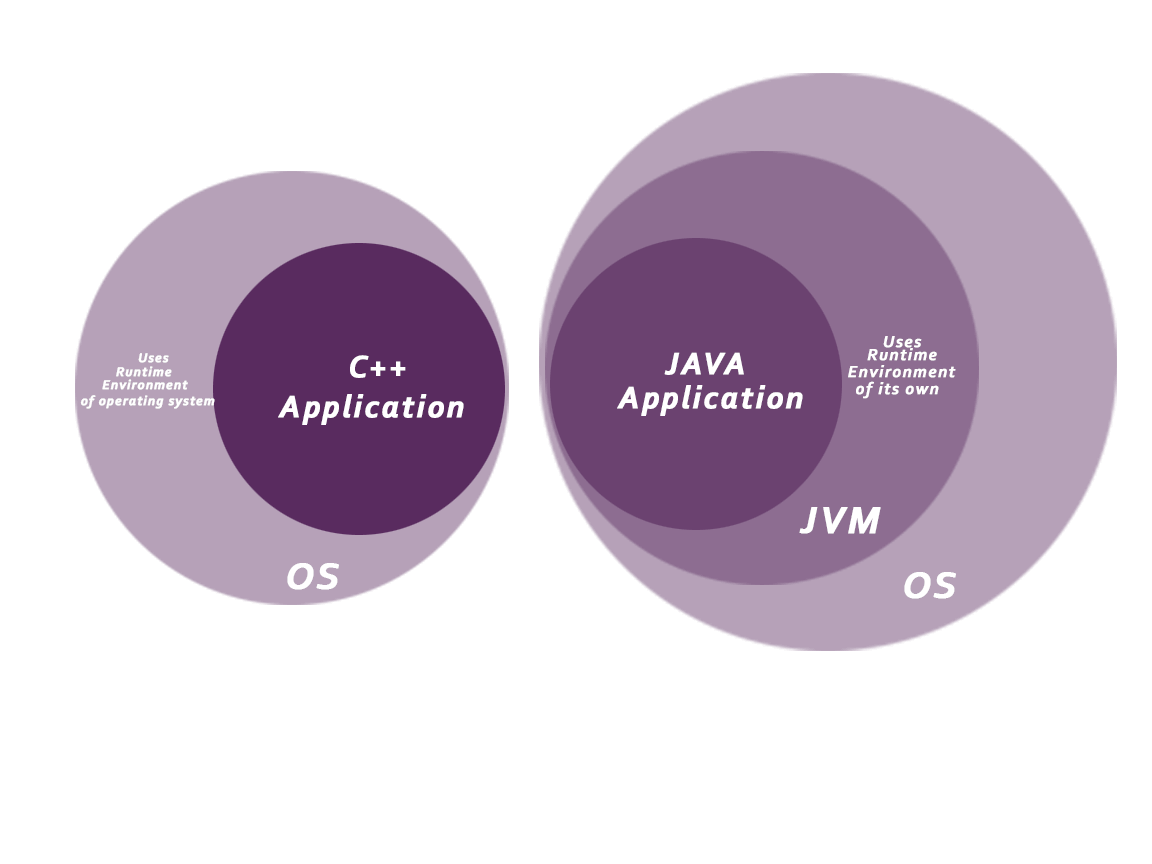
In case of C or C++ (languages that are Platform dependent). The compiler generates an Executable file which is Operating System (OS) dependent. When we try to run this Executable file on other OS it does not run, as it is OS dependent hence it’s not compatible with the other OS.

How to create program in any programming language-

1. With IDE (Integrated development environment) e.g. eclipse. Net beans
2. Without IDE (editor + compiler + lib)



How java is Platform independent: Platform independent language means once compiled you can execute the program on any platform (OS). Java is platform independent. Because the Java compiler converts the source code into bytecode, which is Intermediate Language. Byte code can be executed on any platform (OS) using JVM (Java Virtual Machine) for that platform.



**JVM, JRE and JDK**

**JVM :** JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

The JVM performs the following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

**JRE:** JRE is an acronym for Java Runtime Environment It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

**JDK:** JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.



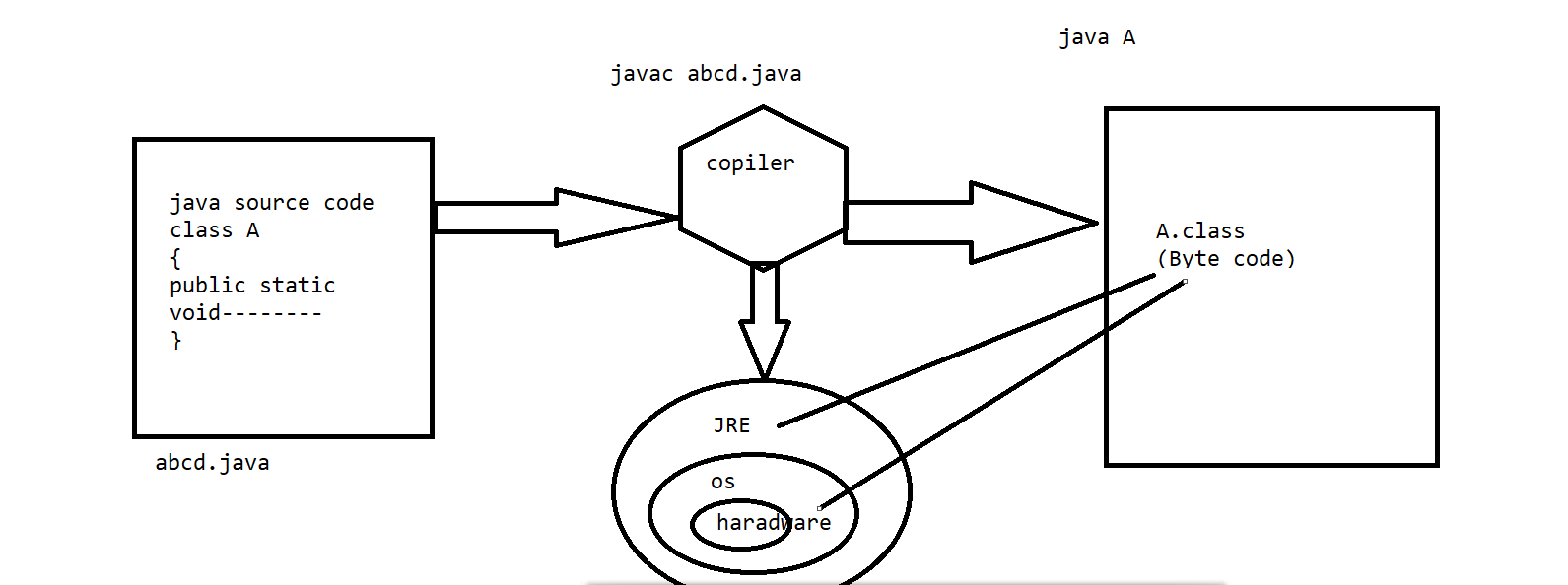
**What is API (Application program interface): -**

Set of inbuilt libraries or predefined functionality in any language whose form may varies from language to language.

In C and C++ API is present in the form of Header files, in .Net it’s in the form of namespace and DLL files whereas in Java it’s in the form of classes, interface and packages.

**What is Framework:** It is the set of API.

**Bytecode: -** Highly optimized, unique and universal code which is created by java compiler with the help of JRE which is runs by Java Virtual Machine (JVM).



**Various Flavours / Editions of Java:-**

* Java Standard Edition (JSE)
* Java Enterprise Edition (JEE)
* Java TV
* Java Card
* Java Micro Edition (JME)
* Java Embedded

**Various Areas of Java:-**

* [Mobile Applications](https://www.edureka.co/blog/applications-of-java/#mobile)
* [Desktop GUI Applications](https://www.edureka.co/blog/applications-of-java/#gui)
* [Web-based Applications](https://www.edureka.co/blog/applications-of-java/#web-based)
* [Enterprise Applications](https://www.edureka.co/blog/applications-of-java/#enterprise)
* [Scientific Applications](https://www.edureka.co/blog/applications-of-java/#scientific)
* [Gaming Applications](https://www.edureka.co/blog/applications-of-java/#gaming)
* [Big Data technologies](https://www.edureka.co/blog/applications-of-java/#bigdata)
* [Business Applications](https://www.edureka.co/blog/applications-of-java/#business)
* [Distributed Applications](https://www.edureka.co/blog/applications-of-java/#distributed)
* [Cloud-based Applications](https://www.edureka.co/blog/applications-of-java/#cloud-based)

**Features of java:**

1. Simple
2. Secure
3. Platform Independent
4. Architectural Neutral
5. Portable
6. Robust
7. Multithreaded
8. Interpreted
9. Distributed
10. Dynamic
11. High Performance
12. Object Oriented

Features of Java in Details:

**Simple**:

* Java is simple as it supports OOPs and we can say that OOPs supports the real world.so it’s simple to develop program in java by relating the problem with real world.
* It’s syntactically similar to C and C++.
* It doesn’t provide explicit use of pointers.
* It doesn’t have concept of Operator overloading.
* Its Automatic Garbage feature also makes it simpler.

**Secure:**

* It doesn’t has support of Explicit use of pointers which prevents someone to gain the direct access to a memory location .Hence making the program secure.
* It has the facility of bytecode verification using the concept of magic number. The magic number is the part of bytecode.
* An executable file is generated only when we run the program and it’s automatically deleted by the JVM after the program execution.
* Java also has Java Authentication and Authorization Service (JAAS) for *authentication* of users, to reliably and securely determine who is currently executing Java code.
* Java also provides the application level security called encryption and decryption.
* Sandbox security checks the security concern on the network.

Platform Independent: - Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language.

Architectural Neutral: - Software that is designed without regard to the target platform are known as Architectural Neutral Software and the language which provide facility to develop such software is known as Architectural Neutral Language.

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

**Portable: -** The literal meaning of portable is “Easy to carry”. (WORA)

Java is portable as we can carry the bytecode easily to various platform also it is platform independent while C and C++ are only portable they’re not platform independent.

**Robust: -** The literal meaning of robust is “Strong”.

**Java is robust because: -**

* It uses strong memory management. There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
* There are exception handling and the type checking mechanism in Java.
* There is a lack of pointers that avoids security problems. All these points make Java robust.

**Multithreaded: -** The Java platform is designed with multithreading capabilities built into the language. Thread is a process of executing the multiple tasks simultaneously at a time. Its implementation is provided by two sub-concepts.

1. **Process**
2. **Thread: -** It’s a light weight process of a sub-process. A thread is the smallest unit of a processing. It doesn’t have any memory address.

**Interpreted: -**  Java has both Compiled and Interpreter Feature Program of java is First Compiled and Then it is must to Interpret it .First of all The Program of java is Compiled then after Compilation it creates Bytes Codes rather than Machine Language.

Then After Bytes Codes are converted into the Machine Language with the help of the Interpreter so for executing the java Program First of all it is necessary to compile it then it must be Interpreter. JVM is the interpreter of java.

**Distributed: -** Distributed applications (distributed apps) are applications or software that runs on multiple computers within a network at the same time and can be stored on servers or with cloud computing. Unlike traditional applications that run on a single system, distributed applications run on multiple systems simultaneously for a single task or job.

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

**Dynamic: -** “Dynamic” just means changing, which generally (in programming languages) means changing something at run time that isn’t explicitly coded in the source code.

Java is a dynamic language as it supports dynamic loading of classes. It means classes are loaded on demand it allocates memory at runtime for classes, methods and objects. This feature allows programmer to allocate memory when required.

**High Performance: -** Java code is compiled into bytecode which is highly optimized by the Java compiler, so that the Java virtual machine (JVM) can execute Java applications at full speed. In addition, compute-intensive code can be re-written in native code and interfaced with Java platform via *Java Native Interface* (JNI) thus improve the performance.

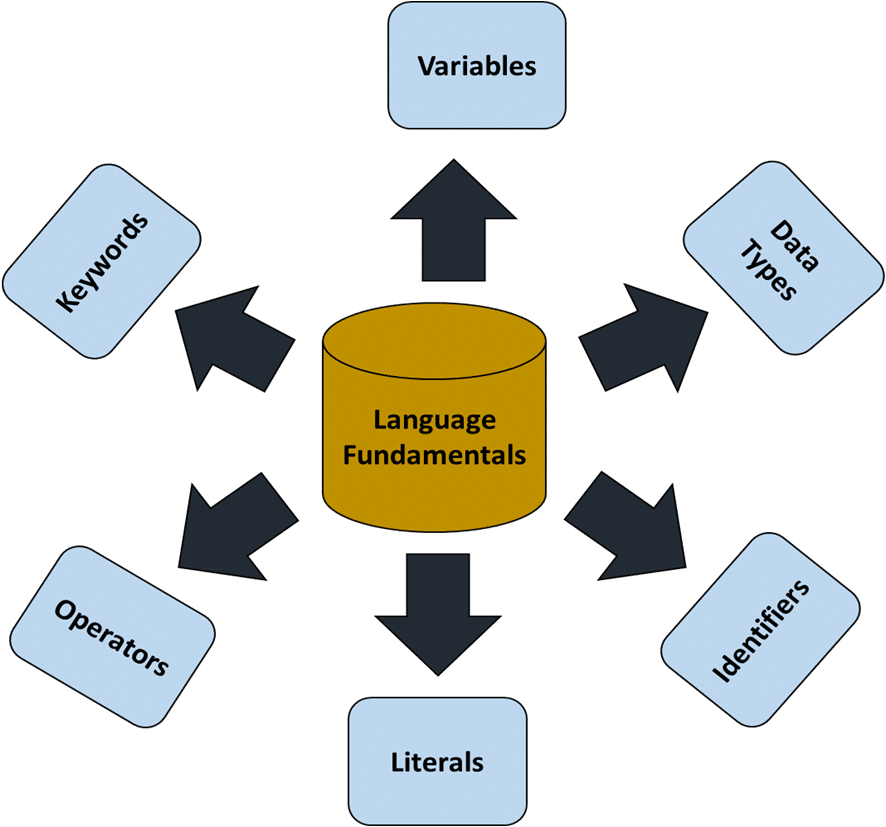
**Object Oriented: -** Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

**Basic concepts of OOPs are:**

* [Object](https://www.javatpoint.com/object-and-class-in-java)
* Class
* [Inheritance](https://www.javatpoint.com/inheritance-in-java)
* [Encapsulation](https://www.javatpoint.com/encapsulation)
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java)

**JAVA FUNDAMENTALS**



1. Keywords: - A keyword is a predefined reserved word whose meaning is fixed in the programming language.

**Type of keywords in java: -**



**There are approximately 52+ keywords used in java.**

\*Java is a Case Sensitive language.

In java all keywords must be written in small letters of Lower Case.

1. **Identifiers: -** Identifiers are the names of variables, methods, classes, packages, object, label and interfaces.

**Rules to declare Identifiers: -**

1. It can’t be a keyword.
2. The length of an identifier must be greater than 0 means it can’t be blank.
3. The first character must be picked from: alpha, underscore, or dollar sign. The first character cannot be a digit.
4. Blank Space and dot operator are not allowed.
5. In special characters only ‘\_’ Underscore and ‘$’ symbols are allowed.

* **Examples of legal identifiers:** age, $salary, \_value, \_\_1\_value.
* **Examples of illegal identifiers:** 123abc, -salary.

**Conventions to Declare Identifiers:**

**Conventions for class identifiers:-**

* If a class identifier contains single word, the 1st letter of the class-name should be in capital letters because the inbuilt classes made by Sun Microsystems have the 1st letter of their class-names in capital.

The main advantage of this convention is that the readability of the program increases.

* If a class identifier contains more than one word, then the 1st letter of each word should be capital letters.
* #Example

**public** **class** **Employee public class EmployeeDetails**

{   {

//code snippet   //code snippet

}   }

**Conventions for interface identifiers:-**

* The convention for interface identifiers is same as that of the class identifiers. Should be capital letters.

#Example

**interface** **Printable interface PrintableInter**

{   {

//code snippet } //code snippet   }

**Conventions for method identifiers:-**

* If a method identifier contains a single word, then it should be written in lower case.
* If a method identifier contains more than one word, the 1st letter of each word should be capital except the 1st word.

**class** Employee   **class** Employee

{   {

//method   //method

**void show()   void showDetails()**

{   {

//code snippet   //code snippet

}   }

}   }

**Conventions for variables identifiers or data members:**

* All the variables are written in lower case.
* All private and local variables use only lowercase letters combined with underscores.
* Variables that represent constant values use all uppercase letters and underscores between words.

#Example

**public** **class** **Employee**

{

int id;

String college\_name;

float price1;

int \_rn;

int $mob;

pubic static final int MAX\_AGE;

}

**Conventions for package.**

All the variables are written in lower case.

#Example

**package employeeofsurya;** //name of package

**public class EmployeeDetails**

{

//code snippet

 }

**Conventions for Constant identifiers:-**

* It should be in uppercase letters such as RED, YELLOW.
* If the name contains multiple words, it should be separated by an underscore ‘\_’ such as MAX\_PRIORITY.
* It may contain digits but not as the first letter.

#Example

**class** Employee

{

//constant

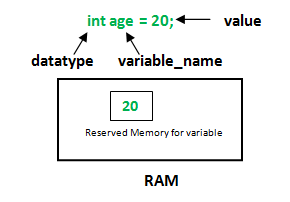
**static** **final** **int** MIN\_AGE = 18;

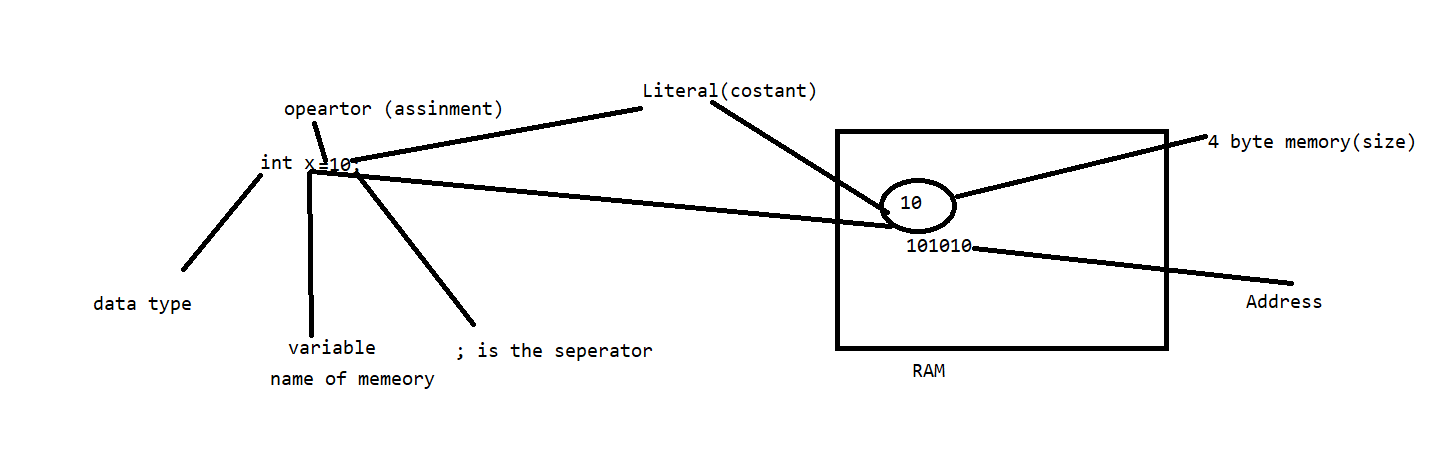
//code snippet

}

**Variable: -** A variable is a name given to a memory location. It is the basic unit of storage in a program which refer the value of specific address.

* The value stored in a variable can be changed during program execution.
* A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
* In Java, all the variables must be declared before use.
* From OOPs point of view in java variables are also known as Data Members (D.M).





**Types of Variables or Data Members in java**

1. Non static variable / Instance variable / Non static data member

* Instance variables are non-static variables and are declared in a class outside any method, constructor or block.
* As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
* Unlike local variables, we may use access modifiers for instance variables. If we do not specify any access modifier then the default access modifier will be used.
* Initialisation of Instance Variable is not mandatory. Its default value is 0
* Instance Variable can be accessed only by creating objects.

2. Static Variable / Class variable / Static data member

* These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
* Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
* Static variables are created at the start of program execution and destroyed automatically when execution ends.
* Initialisation of Static Variable is not mandatory. Its default value is 0
* If we access the static variable like Instance variable (through an object), the compiler will show the warning message and it won’t halt the program. The compiler will replace the object name to class name automatically.
* If we access the static variable without the class name, Compiler will automatically append the class name.
* To access static variables, we need not create an object of that class, we can simply access the variable as ***class\_name.name\_of\_static\_data\_member***

3. Local variable: -

* A variable defined within a block or method or constructor is called local variable.
* These variable are created when the block in entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
* The scope of these variables exists only within the block in which the variable is declared. i.e., we can access these variable only within that block.
* In java there is no concept of garbage value because java itself assign default value to the variables based on the data type.
* Local Variables must be initialized before the first use as they don’t have default value.

#Example



**Data Types: -** It specifies the following things of a variable: -

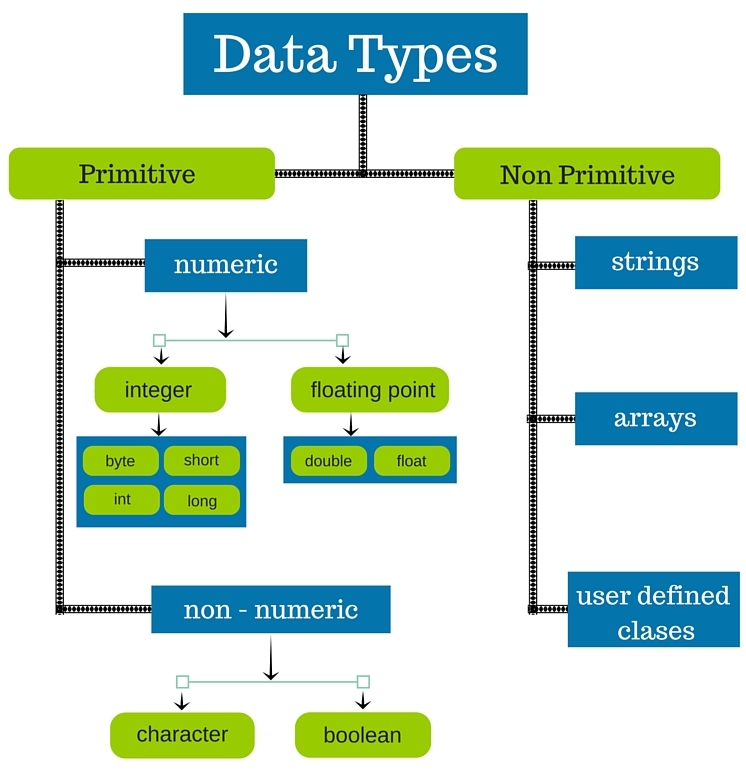
1. Type of value it can hold

2. Size it will occupy in the memory

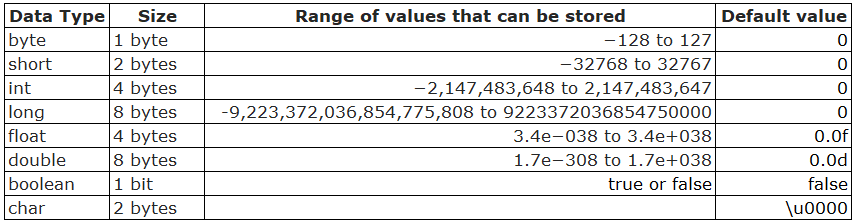
3. Range in which it can hold the data

**There are two types of data types in Java:**

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

**There are 8 types of primitive data types:**

* boolean data type
* byte data type
* char data type
* short data type
* int data type
* long data type
* float data type
* double data type



1 byte = 8 bit

**\*\*Logically there are two types of data types:**

1. **Signed data type (+Ve and -Ve)**

For signed data type, the range can be calculated as -2n-1 to 2n-1-1. Where n is number of bits.

1. **Unsigned data type.(+Ve)**

#*Note: - In java there is no any concept of unsigned data type.*

* **Why the size of char in java is 2 bytes whereas in C and C++ it’s only 1 byte?**

It is because java uses Unicode system not ASCII code system.

# **Unicode System**

|  |
| --- |
| Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages. |

### Why java uses Unicode System?

|  |
| --- |
| Before Unicode, there were many language standards: |
| * **ASCII** (American Standard Code for Information Interchange) for the United States. * **ISO 8859-1** for Western European Language. * **KOI-8** for Russian. * **GB18030 and BIG-5** for Chinese, and so on. |

## **Problem**

|  |
| --- |
| **This caused two problems:**   * **A particular code value corresponds to different letters in the various language standards.** * **The encodings for languages with large character sets have variable length.** * **Some common characters are encoded as single bytes, other require two or more byte.** |

**Solution**

* To solve these problems, a new language standard was developed i.e., Unicode System
* In Unicode, character holds 2 byte, so java also uses 2 byte for characters.

**Lowest Value:** \u0000

**Highest Value:** \uFFFF

**Literals in java: -** Constant values that are typed in the program as a part of the source code are called literals. There are 5 types of literals in java

1. Integral literals
2. Floating point literal
3. Char literal
4. String literal
5. Boolean literals

\*String is a referenced type literal except this all other are primitive literals

**Rules regarding literals: -**

* All String literals are treated as String class object and these must be written in double quotes.
* By default all the integral literals are treated as int data type and floating literals are treated as double data type.
* Character literal is always to be written in single quotes.
* Boolean literals take one of the two values true OR false.

**We can specify floating point literal as float type by suffixed with f or F.**

#Example



**Methods in java**

In Java Methods are also known as Member functions.

A **method** is a block of code which only runs when it is called.

* Methods are used to perform certain actions, and they are also known as **functions**.
* We use methods for the reusability of code. Once a method is defined it can be used again and again.
* A method is a self-contained block.
* A method doesn’t perform any task until it’s called.
* You can pass data, known as parameters, into a method.
* From OOPs point of view a member function represents the behaviour of an object.
* If we make a method public and static, then it acts as a global method thus can be accessed from anywhere.

**Rules Regarding Methods in Java:-**

* Before using / calling a method, we have to define it.
* We don’t have to declare a method in java

int add(); //Not Valid

* It’s valid only with the abstract keyword

abstract int add(); //Valid

**#Syntax to create a method: -**

*Modifiers + Return type + Method Name (Arguments If any)*

*{*

*Body of the method….*

*return value; (if return type is mentioned above)*

*}*

**#Syntax to call a method: -**

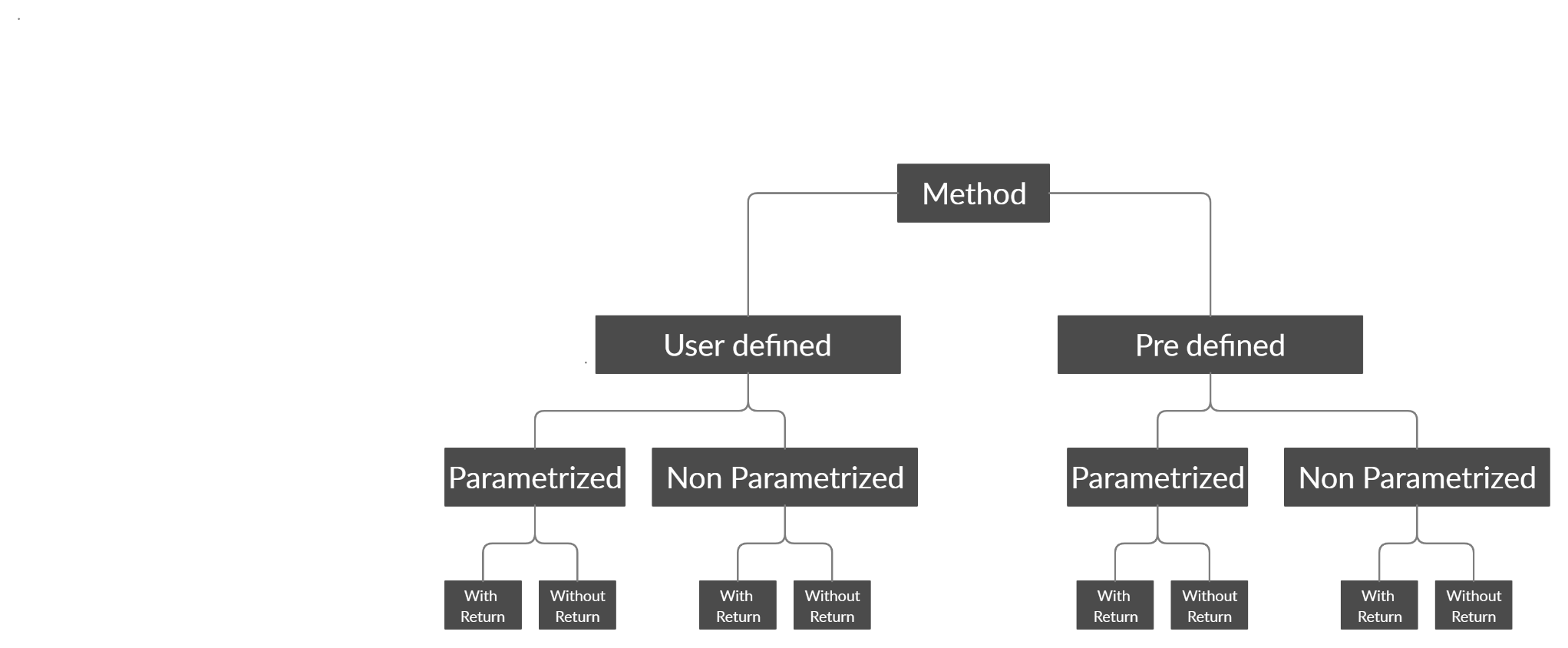
*Method\_name(args if any);*

\*\*We cannot create a method inside another method.

#Example



**Types of Methods**



#Example

* class Test
* {
* static void hello()   // non parameterized without return
* {
* System.out.println("hello");
* }
* static int hello1() // non parameterized with return
* {
* System.out.println("hello");
* return 19;
* }
* static void hi(int x)  //  parameterized without return
* {
* System.out.println("hello");
* }
* static String hi1(int x)  //parameterized with return
* {
* System.out.println("hello");
* return "hello";
* }
* public static void main(String ss[])
* {
* System.out.println("main");
* hello();    }
* }

**Type conversion in java**

It is the process of converting one type of data in another type.

* Primitive to Primitive.
* Primitive to Object.
* Object to Primitive.

**There are two types of Type Conversion: -**

1. Type Promotion (Implicit).
2. Type Casting (Explicit).

**Type Promotion (Implicit): -**

* It is the process of converting one type of data into another type implicitly by java itself.
* If we have data and data does found its exact match so it is implicitly promoted to its higher data type



#Example

* public class Test
* {
* void show(byte b)
* {
* System.out.println("Byte="+b);
* }
* void show(short b)
* {
* System.out.println("short="+b);
* }
* void show(int b)
* {
* System.out.println("int="+b);
* }
* void show(long b)
* {
* System.out.println("long="+b);
* }
* void show(float b)
* {
* System.out.println("float="+b);
* }
* void show(double b)
* {
* System.out.println("double="+b);
* }
* public static void main(String [] args)
* {
* Test t=new Test();
* //t.show(23l);
* //t.show(23);
* t.show(23.67f);
* }
* }

**Type Casting (Explicit): -**

It is the process of converting one type of data into another type explicitly by java programmer.

**#Syntax: -**

***dataType variableName = (dataType) variableToConvert;***

//Java program to illustrate explicit type conversion

class Test

{

    public static void main(String[] args)

    {

        double d = 100.04;

        byte b=20;

        int  c=b;// type promotion

        //explicit type casting

        long l = (long)d;

//explicit type casting

        int i = (int)l;

        System.out.println("Double value "+d);

        //fractional part will  lost

        System.out.println("Long value "+l);

        //fractional part will lost

        System.out.println("Int value "+i);

    }

}

Note: - In java, compiler perform operation only on two type of data either int or double.

Note: - Whenever we perform any arithmetic operation then calculation is done on the basis of below formula:

*Max (type of first operand, type of second operand, type of third operand…)*

Note: - if data type are equals result will be in the same data type.

public class Test

{

    public static void main(String[] args)

    {

        int male=5;

        int female=2;

        float ratio=(float)male/female;

        System.out.println(ratio);

    }

}

Output :- 2.5

**Separators: -** They are symbol which is responsible to separate a method, class, statement, interface, package etc.

1. . (dot)
2. , (comma)
3. ; (semi column)
4. (), {}, [] , :

**Comments: -**

They are also a symbols they are used for ignoring the line as well as for explaining the steps. It increases the readability of code.

1. Single line----- //
2. Multiline -----/ \*--------\*/
3. Documentation----/\*\*-------\*/

The nesting of single line comment is possible. But not possible is multiline.

#Example

public class Test

{

// This is an Example of  single line comment

/\* This is an

    Example of Multiline

    Comment \*/

    /\*\* This is an Example of

documentation comment \*/

}

**Operators: -**

In java programming operators are the symbols which are used to perform the operation over operand.

*int x=10;*

*int y=20;*

*in x+y* ‘+’ is the operator and ‘x’ and ‘y’ are operands.

**Types of Operators: -**

**1. Unary Operators:** The Java unary operators require only one operand.

#Example: Increment Operator (++), Decrement Operator (--), Not Operator (!)

**2. Binary Operators:** The Java binary operators require two operands.

#Example:

* Arithmetic (+,-,\*,/,%)
* Logical (&&, || , !)
* Assignment(=)
* Relational (>,<,>=,<=,!=)
* Bitwise (&,| )
* Shift (>>,<<)

**3. Ternary Operators:** The Java unary operators requires three operands.

Java has only one ternary operator which is also known as Conditional Operator

#Example: Ternary Operator (? : )

**Operator Precedence and Associativity: -**

**Precedence of operators in Java: -** Operator precedence determines the order in which the operators in an expression are evaluated.

**Associativity of Operators: -** If an expression has two operators with similar precedence, the expression is evaluated according to its associativity.

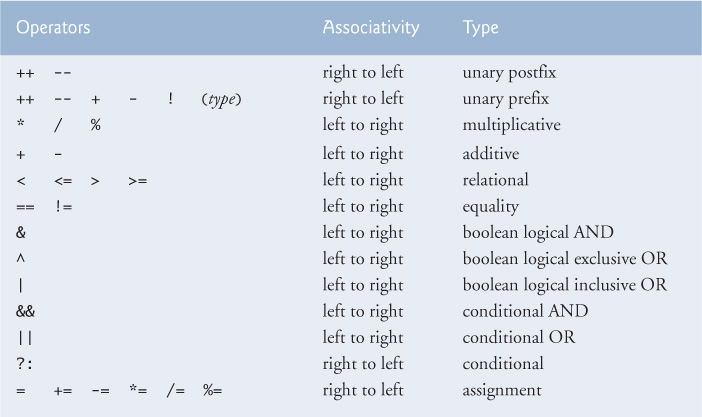
**Associativity is of two types: -**

1. Left to right (grouping is done from left to right).
2. Right to left (grouping is done from right to left).

\*\*Except postfix increment and decrement operators, all the unary operators have right to left associativity.

\*\*Except relational and assignment operators, all the binary operators have left to right associativity.

**Operators Precedence and Associativity Table:**



**What is object oriented programming?**

It is the programming methodology which we can relate to the real world in order to simplify the software development and maintenance.

**This methodology has many concepts like:**

* Polymorphism
* Inheritance
* Encapsulation
* Abstraction
* Class
* Object etc...

Collectively these concepts are called OOPs. (Object Oriented Programming Concepts).

Object: - The term object can be explained in so many ways

* From real world point of view
* Form coding point of view
* From memory point of view
* From class point of view
* Object is any things which exist is the real word which we can see, touch feel & imagine.
* Every real-world entity is called object.
* Every real-world entity which physical logical tangible or intangible existence knows as object.

**Every object has the following three characteristics: -**

1. **State**
2. **Behaviour**
3. **Identity**

#Example

If we have a Mobile phone then its characteristics be like

**State: -** Colour, Price, Dimensions, Camera, RAM, ROM etc.

**Behaviour: -** Calling, Mailing, Messaging, Internet Surfing.

**Identity: -** IMEI Number.

Now we know that every object has, state, behaviour, and identity but in real world we can see that there are many objects in the real world which have same state and behaviour and this is a problem so to overcome this problem we categorize the objects. Thus we find the solution in the form of class of categorization.

Class or Classification: -

* A class is the logical categorization of similar type of object.
* A class is combination of data member and member function.
* A class is the blue print from an object
* A class is the user define data structure.
* A class is user define data type.
* A class is the way to implementing the encapsulation

Class: -A class is the template (blue print) for an object and object is the instance for the class.

#Syntax to create a class in java

*class Identifier*

*{*

*[ Data Member]*

*[Member Function]*

*Special Member Function*

*Constructor*

*[Initializer (init block & static block)]*

*}*

***[ ] symbol indicate that this is optional for the class.***

**Constructor: -** A constructor is a special type of member function which have following characteristics which makes it different from other member function.

1. Its name must be name as the name of class.
2. It doesn’t has any explicit return type not even void.
3. Programmer can’t call it explicitly, it’s implicitly called when object of the class is created using *new* keyword.
4. The main purpose of constructor is to initialize the object at the time of creation.

\*\*If we’re not explicitly creating a constructor in this case compiler implicitly adds a default constructor.

**Types of constructor: -**

1. **Parameterized**
2. **Non-Parameterized**

**Syntax to create constructor: -**

*Access Specifier ClassName(args if any)*

*{*

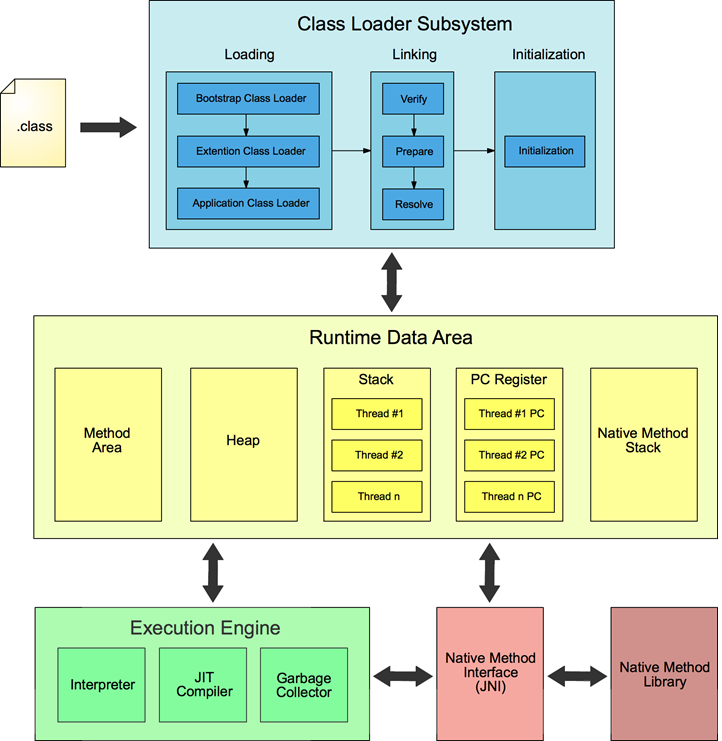
*}*

**Access Specifier**- public, protected, default & private

**JVM Memory Area: -**

1. **Heap Memory Area** (Objects [Instance variables]) – One heap per JVM
2. **Method Area** (method table [contains methods], class area [contains static DM])—One per method per JVM
3. **Stack Area** (Stack Frame [LVA, OS, FD] [Local Variables]) – one stack per thread
4. **PC Register** (One PC register per thread)
5. **Native Method Stack** (Native methods)

**JVM Architecture Diagram**

****

**\*\*Methods are stored in method area but calling of method is done in stack area.**

**#.Who is responsible to call the main method of java.**

#Ans: - JVM is responsible to call the main method of java.

**#Difference between Function and Member function: -**

#Ans. Function represent independent functionality of an application whereas a member function represents the behaviour of an object.

Function can be called independently whereas member function must be called on member (object)

*void morningWalk()*

*{ }*

*morningWalk(); // Function*

*Ram.morningWalk(); // M.F*

**Why we are creating the object of a class:-**

1. Because class does not have its existence in the memory until we create the object of a class.
2. To allocate memory for the instance variable.
3. To call the Non static method.

**How many ways are used to create an object in java:-**

1. By new Keyword.
2. By *newInstance()* method.
3. By Factory Pattern.
4. By clone method.

**Syntax to creating an object in java by new Keyword:-**

*new Class Identifier( args if any); // Anonymous*

*ClassName refvar* ***=****new Class Identifier( args if any);*

*ClassName refvar;*

***r****ef****=****new Class Identifier( args if any);*

//Crating Object Using new .

class Trainee{

new Trainee();

Trainee t=new Trainee();

Trainee t;

t=new Trainee();

}

**Anonymous Object:-**

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only. If you have to use an object only once, an anonymous object is a good approach.

*new* Is the keyword or it is the text based operator whose main aim is to allocate memory in heap are.

**Memory Allocation in Java: -**

1. Static Memory Allocation or Compile time.

2. Dynamic Memory Allocation or Runtime.

In case of static memory allocation the user requirement and identification of memory is decided at compile time.

In case of dynamic memory allocation the user, requirement and identification of memory is decided at runtime.

**\*\*But in both cases memory is allocated at run time.**

When memory is allocated at compile time then it is referred via its name and when memory is allocated at runtime it is referred via pointer.

In Java memory allocation happens only dynamically.

**Reference Id:**

Whenever we are creating an object of a class then a unique identity is crated in memory in the form of 0101- here java Object class methods refer this id and create another id known as reference id whose syntax is-

**ClassName@HexString**

The logic of this reference id is available in Object class toString() method.

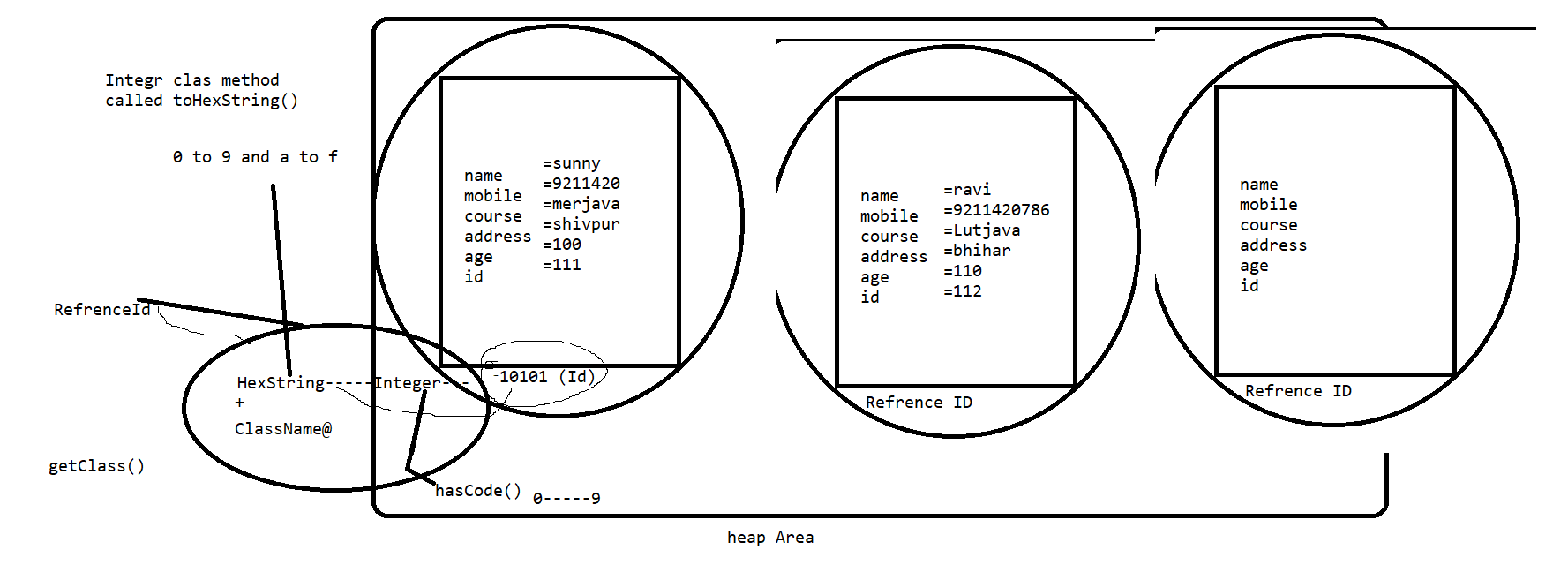
**Reference Variable: -**

Those variables which holds the reference id of an object.

*Trainee t=new Trainee();*

Here **t** is the reference variable.

Single reference id can be hold in multiple reference variables. In other words we can say that different reference variables can point the same object.



#Example

class Trainee

{

String name,mobile,course,address;

int age,id;

void showInfo()

{

System.out.println(name);

System.out.println(mobile);

System.out.println(course);

System.out.println(address);

System.out.println(age);

System.out.println(id);

}

/\*

public String toString()

{

return getClass().getName()+'\*'+Integer.toHexString(hashCode());

}

\*/

void setData()

{

name="ravi";

mobile="79789798";

course="java";

address="grnoida";

age=23;

id=12313;

}

Trainee(String n,String m,String c,String adr,int a,int i)

{

name=n;

mobile=m;

course=c;

address=adr;

age=a;

id=i;

}

Trainee()

{

name="ravi";

mobile="79789798";

course="java";

address="grnoida";

age=23;

id=12313;

}

void setData(String n,String m,String c,String adr,int a,int i)

{

name=n;

mobile=m;

course=c;

address=adr;

age=a;

id=i;

}

void read()

{

System.out.println("trainees r reading....");

}

void chat()

{

System.out.println("trainee r chating.......");

}

public static void main(String ss[])

{

    Trainee t1=new Trainee();

    Trainee t2=new Trainee();

    t1.showInfo();

    t2.showInfo();

    Trainee t3=new Trainee("monu","242424","java","gkp",12,111);

    t3.showInfo();

    Trainee t4=new Trainee("monu","242424","java","gkp",12,111);

    t4.showInfo();

/\*

Trainee t=new Trainee();

t.showInfo();

t.setData();

t.showInfo();

Trainee t1=new Trainee();

t1.setData();

t1.showInfo();

Trainee t2=new Trainee();

t2.setData("monu","242424","java","gkp",12,111);

t2.showInfo();

Trainee t12=new Trainee();

t12.setData("monu1","2142424","1java","1gkp",121,1111);

t12.showInfo();

\*/

//new Trainee().name="manoj";

//Trainee t2;

//System.out.println(new Trainee().name="sunny");

//System.out.println(new Trainee().mobile="9211420");

//new Trainee().chat();

//Trainee t=new Trainee();

//System.out.println(t);

//t.chat();

//t.read();

//t.id=111;

//t.name="sunny";

//t.address="shivpur";

//t.age=100;

//t.course="merjava";

//t.mobile="9211420";

//t.showInfo();

//

//

//Trainee t1=new Trainee();

//System.out.println(t1);

//t1.id=112;

//t1.name="ravi";

//t1.address="bhihar";

//t1.age=110;

//t1.course="Lutjava";

//t1.mobile="9211420786";

//t1.showInfo();

//

//Trainee t3;

//t3=new Trainee();

//System.out.println(t3);

//t3=t;

//System.out.println(t3);

//System.out.println(t);

//t2=t1;

//t3.name="kabi";

//t.showInfo();

}

}

**Copy Constructor: -**

The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The copy constructor is used to − Initialize one object from another of the same type. Copy an object to pass it as an argument to a function.

//Program to demonstrate how copy constructor works

public class Student

{

    String name;

    int age;

    Student(String n,int a)

    {

        name=n;

        age=a;

    }

    Student(Student s)  //copy constructor

    {

    name=s.name;

    age=s.age;

    }

    void show()

    {

        System.out.println(name);

        System.out.println(age);

    }

    public static void main(String[] args)

    {

        Student s=new Student("ram",12);

        s.show();

        Student s1=new Student(s);

//      s1.name=s.name;

//      s1.age=s.age;

        //System.out.println(s1.name);

        s1.show();

    }

}

**Control Statements in java: -**

Control Statements are used to control the execution flow of the program.

**There are three types of control statements:**

1. **Conditional Control Statements**

### a) If statement: It will go inside the block only if the condition is true otherwise, it will not execute the block.

if(condition){

    // statements (if Block)

    }

    //other statements

**b) If statement:** If the condition is true then, it will execute the If block. Otherwise, it will execute the Else block.

if(condition){

    // statements (if Block)

    }

    else{

    // statements (Else block)

    }

    //other statements

### c) If Else-If statement: if the condition is true, then it will execute the If block. Otherwise, it will execute the Else-If block. Again, if the condition is not met, then it will move to the else block.

### d)  Switch Statement: - Switch statement allows program to select one action among multiple actions during the program execution.

### #Example

class WriteExample{

    public static void main(String []args){

    int a=2; int b=3; int c=4;

    if(a>b){

    System.out.print(“Institute1”);

    }

    else if(a>c){

    System.out.print(“Institute2”);

    }

    else{

    System.out.print(“AndroJava”);

    }

    }

    }

OUTPUT: - AndroJava

Switch(variable/value/expression){

    Case :

    // statements

    Case :

    // statements

    default:

    // statements

    }

1. **Looping Control Statements**

These are used to execute a block of statements multiple times. It means it executes the same code multiple times so it saves code. These are also called Iteration statements.

**a) for Loop**

* It executes the code until condition is false.
* It is used when number of iterations are known.

**#Syntax**

for(initialization; condition; increment/decrement){

    //statements (For Body)

    }

**#Example: -**

class WriteExample{

    public static void main(String []args){

    int a=1;

    for(int i=0; i<2; i++){

    System.out.println(“Value of a is:”+ a); // string concatenation

    a++; // increases value by 1

    }

    }

    }

Output: Value of a is:1

Value of a is:2

**b) while Loop**

* While loop executes till the condition becomes false.

#Syntax: -

while(condition){

    // statements

    }

#Example: -

class AndroJava {public static void main(String args[]) {

    int i = 2;

    while( i <=2 ) {

    System.out.println(“AndroJava” );

    i++;

    }

    }

    }

Output: - AndroJava

**c) do while Loop**

* When you are using for or while, then it will execute the loop body only if the condition if true.
* In do-while loop, it will execute the loop first, then it checks the condition. So, it will execute the loop at least once.
* It is called exit controlled loop while for & while loop are called entry controlled loop.

**#Syntax: -**

do{

    // statements

    }while(condition);

**#Example: -**

class AndroJava {public static void main(String args[]){

    int a = 2;

    do{

    System.out.println(AndroJava);

    i++;

    }while( a< =2 );

    }

    }

1. **Unconditional Control Statements/Jump Statements**  
   **a) Break Statement: -**break is a keyword. It is used within any control statements. It is used to terminate the execution of the current loop or switch statements.

**#Example: -**

class AndroJava{ // break without label

    public static void main(String args[]){

    for(int i=1;i<=4;i++){

    if(i==3) break;

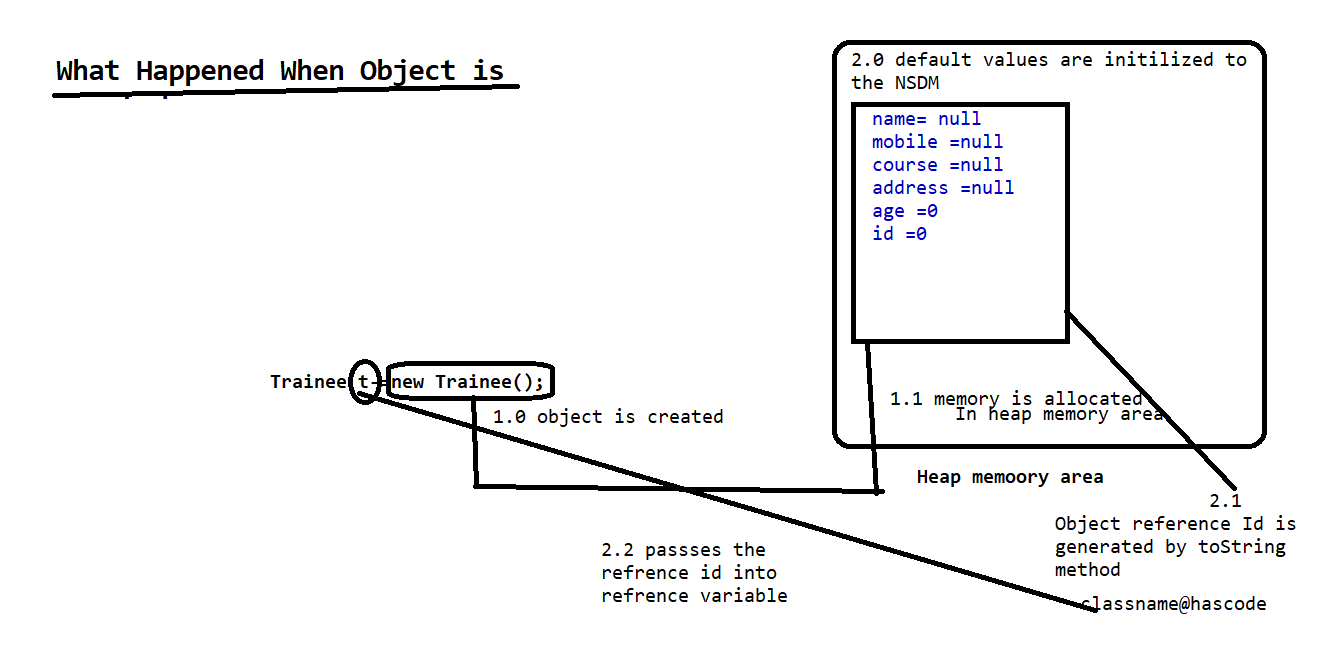
    System.out.println(“AndroJava”);}

    } }

Output: -AndroJava

AndroJava

**What happens when Object is created: -**



**Initializers: -**

An initializer is a line of code (or a block of code) placed outside any method, constructor, or other block of code. Initializers are executed whenever an instance of a class is created. Initializers are used to initialize data even before the constructor.

**Depending upon the behaviour these are of two types:** -

**1.** Init block (Non-Static Initializers) – Object Dependent

**2.** Static block (Static Initializers) – Object Independent

*Syntax to create Init block: -*

*{*

*//Business Login*

*}*

**Features of Init block: -**

1. It executes with respect to every constructor (object).
2. We can have more than one init block in a program in this case the order of execution of init block is determined by the order in which they are written means in First Come First Serve order(FCFS)
3. Init block should be created after declaration of all variables in a program.
4. Never create the same class object in init block but we can create the object of other class in init block as per our requirement.

Note: - In java init block is implicitly used by compiler to initialize every non static data member.

class A

{

int x=10;

}

//During Compilation compiler make the following changes.

class A

{

int x;

{

x=10;

}

}

\*\*The logic of init block is automatically pasted by the compiler at the top of every constructor.

#Example: -

package oops;

public class A

{

    int x;

    static int counter;

    {

    counter++;

    System.out.println("init block1");

    }

static public void countObject()

{

System.out.println("Total number of object="+counter);

}

    A()

    {

        x=10;

        System.out.println("A cons...="+x);

    }

    A(int x)

    {

        System.out.println(x);

    }

    {

        System.out.println("init block2");

        }

    public A(int i, int j)

    {

        System.out.println(i);

        System.out.println(j);

    }

    public static void main(String[] args)

    {

        System.out.println("main block..");

        new A();

        new A(24);

        new A(44,67);

            {

            System.out.println("mai init ka jaisa hu...");

            }

            A.countObject();

    }

        {

        //x=20;

        //System.out.println("init block....="+x);

            System.out.println("init block3");

        }

}

#Example: -

package oops;

public class Employee

{

    String name;

    int age;

    Address  adr;

    Salary sa;

    int x;

    {

      adr= new Address();

    }

    {

      sa=new Salary();

    }

    Employee(String m,int a)

    {

        name=m;

        age=a;

    }

    void show()

    {

    System.out.println("Name="+name);

    System.out.println("Age="+age);

    System.out.println("Address="+adr.city +"State,"+adr.state+" Country,"+adr.country);

    System.out.println(sa.ta +" "+sa.da+" "+sa.hra);

    }

    public static void main(String[] args)

    {

        Employee e=new Employee("ram",34);

        e.show();

    }

}

**Static Keyword in Java: -**

In Java, static keyword is mainly used for memory management.

**Static keyword can be used with:** -

1. Data member
2. Member function
3. Block
4. Nested class.

**Static keyword can NOT be used with:** -

1. Class
2. Constructor
3. Interface
4. Local variable.

**#Example Without static: -**

package androjava;

public class Placement {

    String companyname = "HCL";

    String collegename = "Kalinga University";

     String s\_name;

     int s\_id,s\_age,s\_sal;

    Placement(int sid,String sname,int sage,int ssal)

    {

        s\_id=sid;

        s\_name=sname;

        s\_age=sage;

        s\_sal=ssal;

    }

    void show(){

        System.out.println("\nCollege Name : "+collegename);

        System.out.println("Company Name : "+companyname);

        System.out.println("Student ID : "+s\_id);

        System.out.println("Student Name : "+s\_name);

        System.out.println("Student Age : "+s\_age);

        System.out.println("Student Monthly Salary : "+s\_sal);

    }

    public static void main(String args[])

    {

    System.out.println("\*\*\*\*\*Kalinga University Student Placement\*\*\*\*\*");

    Placement p1=new Placement(9211,"Amit",24,40000);

    Placement p2=new Placement(9233,"Sumit",25,30000);

    Placement p3=new Placement(9243,"Ronit",28,45000);

    System.out.println();

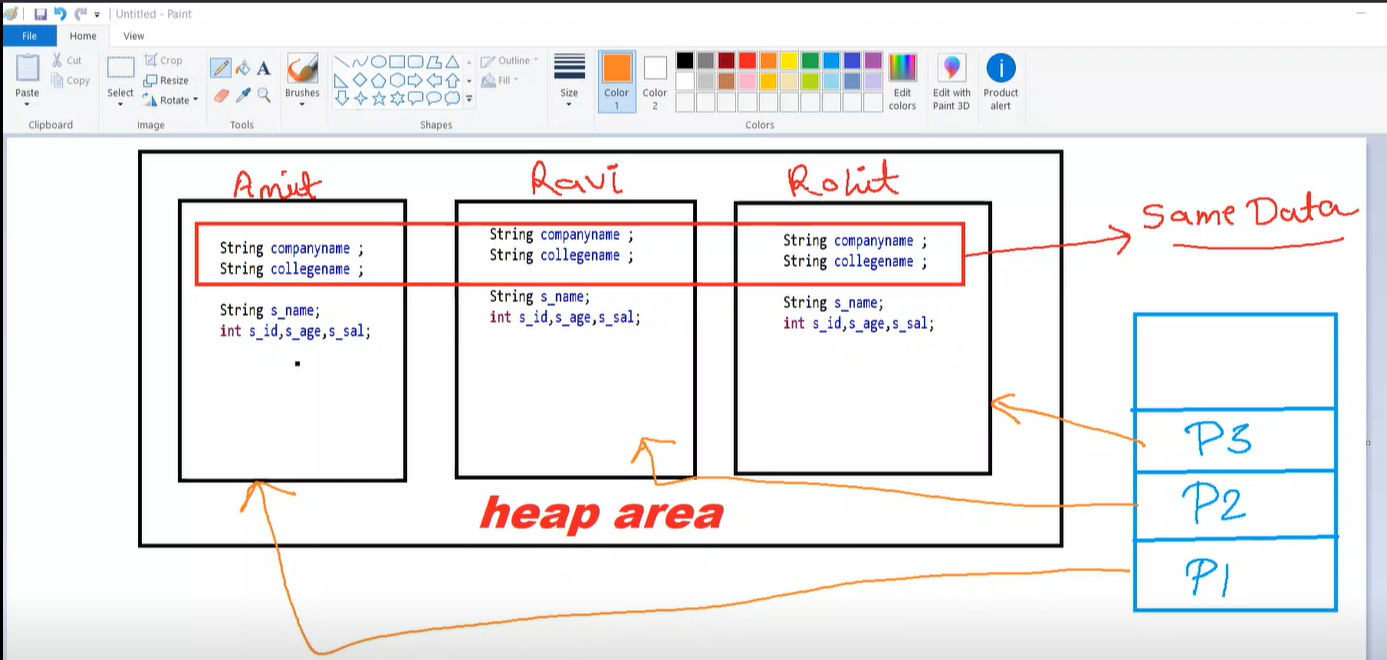
    p1.show();

    p2.show();

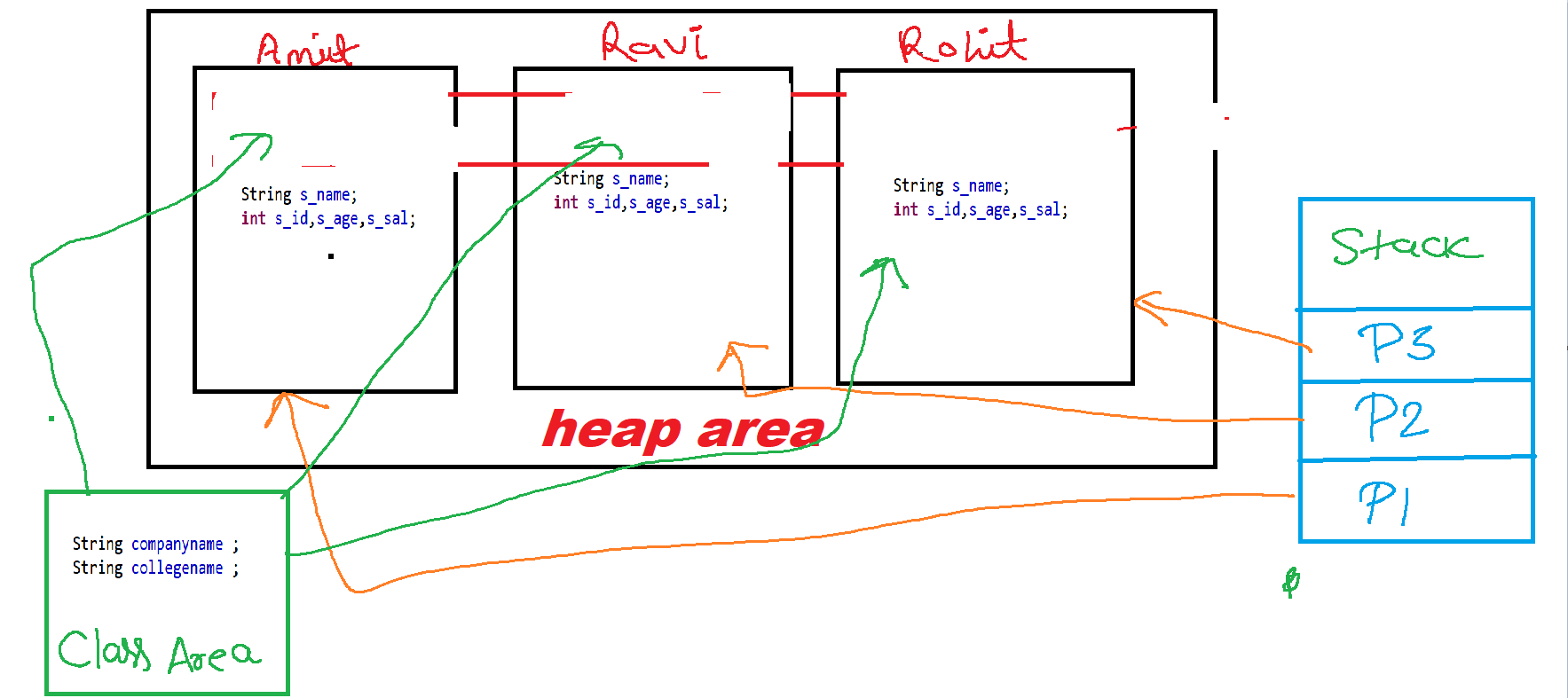
    p3.show();

    }

}



**#Example With static: -**



package androjava;

public class Placement {

    static String companyname = "HCL";

    static String collegename = "Kalinga University";

     String s\_name;

     int s\_id,s\_age,s\_sal;

    Placement(int sid,String sname,int sage,int ssal)

    {

        s\_id=sid;

        s\_name=sname;

        s\_age=sage;

        s\_sal=ssal;

    }

    void show(){

        System.out.println("\nCollege Name : "+collegename);

        System.out.println("Company Name : "+companyname);

        System.out.println("Student ID : "+s\_id);

        System.out.println("Student Name : "+s\_name);

        System.out.println("Student Age : "+s\_age);

        System.out.println("Student Monthly Salary : "+s\_sal);

    }

    public static void main(String args[])

    {

    System.out.println("\*\*\*\*\*Kalinga University Student Placement\*\*\*\*\*");

    Placement p1=new Placement(9211,"Amit",24,40000);

    Placement p2=new Placement(9233,"Sumit",25,30000);

    Placement p3=new Placement(9243,"Ronit",28,45000);

    System.out.println();

    p1.show();

    p2.show();

    p3.show();

        }

}

**Static Data Member and Non static Data member: -**

1. Static data member represents the property of class whereas non-static data member is representing property of object.
2. In case of static data member memory is allocated in class area at class loading time only once and this memory is shared by multiple objects. It means single copy is created in memory in case of static data member.
3. In case of non-static data member memory is allocated in each objects separately (In heap area).It means separate copy is created in each objects in case of non static data member.
4. If a data is common for all objects then make it static otherwise non static.
5. As we know that static represents the property of class that’s why the standard approach to call the static data member is with class name. and non static with object reference

**#Example how memory is shared in case of static data**

package oops;

public class B {

    static int x;

    B()

    {

        x++;

        System.out.println(x);

    }

    public static void main(String[] args)

    {

    new B();

    new B();

    new B();

    }

}

OUTPUT: -

1

2

3

**How Many ways are used to call Non static data member:-**

1. Object.Nonstatic Data member; (e.g. new A().x)
2. Object Reference. Non static Data member (e.g--- a.x)------- (Standard approach)
3. Directly Non static Data member (but only in same class non static member function and constructor ) (e.g .x)

\*\*Above two approaches are valid in same class as well as different class.

**How Many ways are used to call static data member:-**

* 1. Object . static Data member; (e.g. new C().x)
  2. Object Reference. static Data member (e.g--- a.y)-------
  3. ClassName.Staticdatamember.(Standard approach)
  4. Directly static Data member (same class any where) (e.g .y)

\*\*Above three approaches are valid in same class as well as different class.

**Rule: -** A non-static data member or non-static member function cannot directly access in static context (static method or static block) but vice-versa is allowed.

**Static Member Function and Non static member function:-**

1. Static Member Function represents the behaviour or functionality of class and Non static member functions representing behaviour of object.
2. Method is either static or non-static it gets memory in method area (method table).
3. As we know that static represent the behaviour of class that’s why the standard approach to call the static member function with class name and non-static with object reference

**How Many way are used to call Non static member function:-**

1. Object. Non static member function; (e.g. new A().show())
2. Object Reference. Non static member function(e.g--- a.show())------- Standard approach
3. Directly member function(but only same class non static member function and constructor ) (e.g .x)

Above two approaches are valid in same class as well as different class.

**How Many way are used to call static member function:-**

1. Object .static member function; (e.g. new C().disp())
2. Object Reference. static member function(e.g--- a.disp())
3. ClassName. static member function. Standard aproach
4. static member function(same class anywhere) (e.g .disp())

Above three (1 to 3) approaches are valid in same class as well as different class.

**Static Block: -**

1. Are those blocks which executes only once at class loading time.
2. This block is object independent.
3. It execute before main() method.
4. We can write any logic in static block which we can write in main().
5. In case of more than one static block their order of execution in FCFS.

Note: - we can’t pass the command line argument in static block this could be only possible with main method.

//Static Block

static

{

//coding…..

}

**Class loading: -**

It is the process of storing the class specific information into memory (i.e. data member, member function, constructor etc.) this process is done by class loaders. (They are the software program).

**Type of Class Loaders: -**

1. bootstrap class loader**(rt.jar)**
2. extension class loader **(jre/lib/ext)**
3. Application class loader **(environment variable**)

**Things done at class loading time:**

* Memory is allocated to the Static Data member.
* Default values are initialized to the Static Data Member.
* Static Block are executed in given order

**Types of Class Loading:**

1. Static class loading
2. Dynamic class loading.

In case of static class loading compiler have already knowledge about loaded class but in case of dynamic class loading compiler does not have the knowledge about loaded class.

**Condition of static class loading: -**

1. When we run any class.
2. When we are creating the object of a class into another class.
3. When we are inheriting one class into another class.
4. When we call any static members of a class.

**Condition of static class loading: -**

In java there is a class whose name is **Class** and this class **Class** have a static method called **forName()** which is responsible for dynamic class loading?

**Command Line Arguments: -**It is the process of passing the values from command line at run time.

In java there is a class whose name is Class. Every class is the instance of class Class.When any class statically or dynamically loaded into memory then JVM creates object of class Class for the loaded class. Then class Class object is the duplicate copy (reflected image) of specific loaded class. It is the part of reflection API.

**How many ways are there to get the object of class Class:**

1. When we call the getClass() method of object class on the target class object.

Class A

{

Public static void main(String ss[])

{

A a=new A();

Class c=a.getClass():;

}

}

1. When we load a class dynamically using the static method forName() of class Class then it return the object of class Class.

Class c=Class.forName(args[0]);

1. Directly.

Class ca=String.class; //directly

**About forName() method: -**

1. This is the static method of class Class.
2. There are mainly two uses of this method
3. To load a class dynamically
4. Return the class Class object of the loaded class.

//signature of forName()

Public static Class forName(String nameOfClass)

**About newInstance() method: -**

1. This is the non-static method of class Class whose main purpose is to create the object of any class at runtime.
2. This is another way to create the object of a class.

**Wrapper Classes: -**

* Wrapper classes are those classes which provide the facility to convert the primitive data type into object and object to primitive.
* In java there are so many wrapper classes are available for every primitive data types (java.lang)



**Boxing and Unboxing in Java: -**

* The process of converting any primitive data type into object is called boxing and its reverse process is called unboxing.
* From JDK1.5 there is no requirement to convert any primitive data type into object because of the new concept of **AutoBoxing and AutoUnboxing.**

**Explanation of main()**

In java main() method is the method which maintains the entry point of a java program main() is neither predefined nor user defined but it’s a contract between the user and the programming language.

Its Syntax must be like this:

public static void main(String args[])

{

}

\*\*\**We can also use some other modifiers for main like strictfp, final synchronized ,etc. but is must have public ,static, void and a string array argument*

We can change the name of the string array argument. args[] to anything[].

It is used to hold the command line arguments in the form of string values.

In java main() method is run by JVM

**Why main() method is public ?**

public is an access specifier, the members which are public can be accessed from anywhere in the program(outside class ,package). Thus is used before main() so that JVM can access and run the main() method.

**Why main() is static ?**

As we know that Non-Static Method requires instance/object of the class to access it.

But Static method can be accessed without any instance or reference of the class (object).

When JVM starts, there is no object of the class present.

Thus to make sure that JVM is able to run the main() without object it’s made static.

**Why the return type of main() is void ?**

Java main method doesn’t return anything, that’s why its return type is void.

And it was made so by the java developers because JVM doesn’t require any return value from the main() to perform any kind of operations so to keep java simple they made main() as void .

**Overloading of main()**

We can overload main() with different types of parameters.

But JVM will execute main() which have String array as an argument and the overloaded function behaves as a normal function.

classTest

{

    publicstaticvoidmain(String[] args)

    {

        System.out.println("Main Method String Array");

    }

publicstaticvoidmain(int[] args)//behave as a normal method

    {

        System.out.println("Main Method int Array");

    }

}

**Data Shadowing: -** When in a program the name of instance variable and the name of local variable are same this concept is called data shadowing.

In other words data shadowing is the process of having same name of instance and local variable in a program.

package mydata;

public class Demo

{

    int  x=10;

    void dip(int x)

    {

        //int x=20;

        System.out.println(x);

        System.out.println(x);

    }

    public static void main(String[] args) {

        Demo de=new Demo();

        de.dip(20);

    }

}

**\*\*Whenever Data Shadowing is done the preference always goes to the local data members**

package mydata;

public class Demo

{

    int x=10;

    void dip(int x,Demo de)

    {

        //int x=20;

        System.out.println(de.x);

        System.out.println(x);

    }

    public static void main(String[] args) {

        Demo de=new Demo();

        de.dip(20,de);

    }

}

**this keyword in java: -**

* It is the implicit pointer
* It is the implicit reference variable which holds the reference id of the current object.
* This implicit reference is available implicitly in every non static method and constructor

\*\*This keyword can’t be used in static context. (Static method or Block)

Whenever we are calling any non-static method and constructor in the program so by default is method and constructor have an extra argument (i.e. Object of same class) passed by java compiler its self.

Signature of this *public final this;*

**Use 1.** It is used to identify the instance variable in case of data shadowing.

**#Example: -**

//use of this in identification of instance variable in case of data shadowing

Class A

{

Void show(this)

{

}

A(this)

{

}

PSVM()

{

A a=new A(a); // Object of same class

a.show(a);   // Object of same class

A a1=new A(1); // Object of same class

a1.show(a1); // Object of same class

}

}

package mydata;

public class Demo

{

    int x=10;

    void dip(int x)

    {

        //int x=20;

        System.out.println(x);

        System.out.println(this.x);

        System.out.println("This ID="+this);

    }

    public static void main(String[] args)

    {

        Demo de=new Demo();

        System.out.println("object1 Id="+de);

        de.dip(23);

        Demo de1=new Demo();

        System.out.println("object2 Id="+de1);

        de1.dip(20);

    }

}

package mydata;

public class Demo

{

    int x=10;

    Void dip()

    {

    //int x=20; // If local variable not available.

        System.out.println(x);

        System.out.println(x);

    }

    public static void main(String[] args)

    {

Demo de=new Demo();

    de.dip();

    }

}

//Case 1:-If local variable not available…

package mydata;

public class Demo

{

    int x=10;

    void dip()

    {

          //int x=20; // If local variable not available.

        System.out.println(this.x); // this keyword id added J.C

        System.out.println(this.x);

    }

    public static void main(String[] args)

    {

        Demo de=new Demo();

de.dip();

    }

}

//Case 2:-If local variable  available…

package mydata;

public class Demo

{

    int x=10;

    void dip()

    {

          int x=20;

        System.out.println(x); // this keyword not added J.C

        System.out.println(x);

    }

    public static void main(String[] args)

    {

        Demo de=newDemo();

de.dip();

    }

}

Use 2.

This keyword is used as a method argument in order to refer the current class object.

// This keyword is used as a method argument in order to refer the current object.

package mydata;

class B

{

    static void disp(A a)

    {

        a.msg();

        System.out.println("disp");

    }

}

class A

{

    void show()

    {

        B.disp(this);

        System.out.println("show");

    }

    void msg()

    {

        System.out.println("msg");

    }

}

public class ThisTest2

{    public static void main(String[] args)

    {

        A a=new A();

        a.show();

    }

}

Use 3.

This keyword can also be used in case of method chaining.

Method chaining is the process of calling multiple methods on same object at same time.

Note:-In java every constructor have implicit return type i.e class type and it return the reference id of same class in the form of this.

class A

{

}

//Default Constructor

class A

{

A A()

{

return this;

}

}

package mydata;

public class ThisTest3

{

    ThisTest3 disp()

    {

        System.out.println("disp");

        return this;

    }

    void show()

    {

        System.out.println("show");

    }

        ThisTest3()

    {

        System.out.println("cons....");

    }

    public static void main(String[] args)

    {

        new ThisTest3().disp().show();

    }

}

Use 4.

This keyword can also be used in case of constructor chaining.

Constructor chaining is the process of calling multiple constructor in a chain on single object at same time.

Syntax to call constructor: - *this(args if any);*

*this(args if any);* statement must be the very first statement in constructor body in case of constructor chaining.

// Java program to illustrate Constructor Chaining

// within same class Using this() keyword

class Temp

{

    // default constructor 1

    // default constructor will call another constructor

    // using this keyword from same class

    Temp()

    {

        // calls constructor 2

        this(5);

        System.out.println("The Default constructor");

    }

    // parameterized constructor 2

    Temp(int x)

    {

        // calls constructor 3

        this(5, 15);

        System.out.println(x);

    }

    // parameterized constructor 3

    Temp(int x, int y)

    {

        System.out.println(x \* y);

    }

    public static void main(String args[])

    {

        // invokes default constructor first

        new Temp();

    }

}

Use 5.

this keyword can also be used to call the same class methods static and non-static both.

this.show();

show();

**Use 6.**

this keyword can also be used for forward referencing .

A forward reference is a reference to a variable that has not yet been initialized. These are bad simply because, if allowed, they'd give us unexpected results. Take a look at this bit of code:

package mydata;

public class ThisTest4

{

    int x=10;

    int z=x\*this.y;

    int y=20;

    ThisTest4()

    {

    System.out.println(z);

    }

    void show()

    {

        System.out.println("show");

    }

    void disp()

    {

        this.show();

        System.out.println("disp");

    }

    public static void main(String[] args) {

    new ThisTest4().disp();

    }

}

**Encapsulation**

* It is the process of wrapping the data member and member function into a single unit called class.
* The purpose of encapsulation is to achieve data hiding.
* And class is the way to achieve the encapsulation.
* But the normal class is not providing the 100 % implementation of encapsulation.
* Java Bean class providing the best implementation of encapsulation.

**Java Bean class or (POJO) Plain Old java Object:**

A java bean is the general purpose reusable piece of code that must have the following characteristics.

Any normal class said to be java bean class if it has the following characteristics: -

1. This class must be public.
2. This class must be inside package.
3. The data member of the class must be private.
4. This class must not be extends any specific class and implements any specific interface but must implement serializable interface.
5. This class must have public setter and getter method.
6. This class must have public default constructor.

**Note**: - Java bean must implement ***serializable*** interface whereas POJO may or may not.

#Example: -

package com.youtube.login;

public class Login implements java.io.Serializable

{

private String name,pass;

public void setName(String name)

{

this.name=name;

}

public String getName()

{

return name;

}

public void setPass(String pass)

{

this.pass=pass;

}

public String getPass()

{

return pass;

}

}

**Inheritance**

It is a process of acquiring the properties and behaviour of Parent class (Base class/Super class) into a new class called Child class (Sub-class/Derived class) with own properties and behaviour.

In java we use ***extends*** keyword to inherit a class.

**Advantages of Inheritance:-**

1. The purpose of inheritance is reusability.
2. To achieve run time polymorphism.
3. Method overriding.
4. Data Hiding.

#Example: -

package inheritance;

class Base  //This is the Parent / Super / Base class

{

    intx=10;

    staticinty=100;

    static void sMethod() //static method

    {

        System.out.println("Static method "+y);

    }

    voidn sMethod()  //non-static method

    {

        System.out.println("Non StaticMethod"+x);

    }

}

package inheritance;

class Base  //This is the Parent / Super / Base class

{

    int x=10;

    static int y=100;

    static void sMethod() //static method

    {

        System.out.println("Static method "+y);

    }

    void nsMethod()  //non-static method

    {

        System.out.println("Non StaticMethod"+x);

    }

}

public class Child extends Base  //This is child /Sub/Derived class

{

    public static void main(String[] args) {

        System.out.println(new Child().x);

        Child b=new Child();

        System.out.println(b.x);

        new Child().nsMethod();

        b.nsMethod();

        System.out.println(new Child().y);

        System.out.println(b.y);

        System.out.println(y);

        new Child().sMethod();

        b.nsMethod();

        sMethod();

        Child.sMethod();

    }

    void nsm()   //non static M.F of Child class

    {

        System.out.println(x);

        nsMethod();

        sMethod();

    }

    Child ()  //D.C of Child Class

    {

        System.out.println(x);

        nsMethod();

        sMethod();

    }

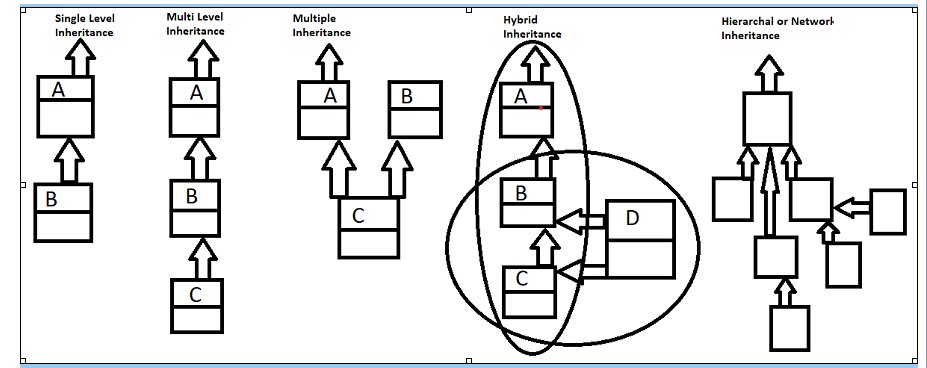
}

**Types of Inheritance: -**

1. Single-Level
2. Multi-Level
3. Multiple
4. Hybrid
5. Hierarchical OR Network Based Inheritance

**\*\*\* In java there is no any class created without inheritance**

* If there’s class name as A then it will extends Object class. This Object class is extended by each and every class in java. Or we can say that object class is the Parent/Super class of all the user defined and pre-defined classes.
* Object class has 9 methods known as magic method.
* Thus we can say that Single level inheritance is by default present in java.

**

*Why java doesn’t supports Multiple Inheritance in case of class: - because it leads to data ambiguity.*

Hybrid Inheritance is the combination of any two inheritances.

**What is Data Hiding?**

* When in a program two classes (parent and Child) have data member with same then this concept is called data hiding.
* In case of data hiding always preference goes to child class data.

package reusability;

class Base

{

    int x=10;

}

public class Child extends Base

{

    int x=20;

    void show()

    {

        System.out.println(x);

        System.out.println(super.x);

    }

    public static void main(String[] args)

    {

        Child c=new Child();

        c.show();

    }

}

\*\*Note: - Data shadowing has done is one class whereas data hiding happens in two classes.

Question:-Now the question is how to print the data of parent class.

Answer: - By creating the object of parent class or by using the ***super*** keyword.

**Super Keyword: -**

* It is the keyword which can be used in so many ways.
* It is used to identify the parent class data in case of data hiding.
* It is used to call the parent class method in case of method overriding.
* It is used to call the parent class constructor in case of constructor chaining.

**Note: - Super can’t be used in static context like this.**

**Up-casting and Down-Casting: -**

* In java the object of child class can be hold into the reference variable of parent class this concept is called up casting.
* But we can’t hold the object of parent into child.

Base b=new Child(); // [valid]

Child c=new Base(); // [Invalid]

Base  c=new Child(); //up casting

Child c1=(Child)c;  // down casting

**Note:-Without up casting down casting is not possible.**

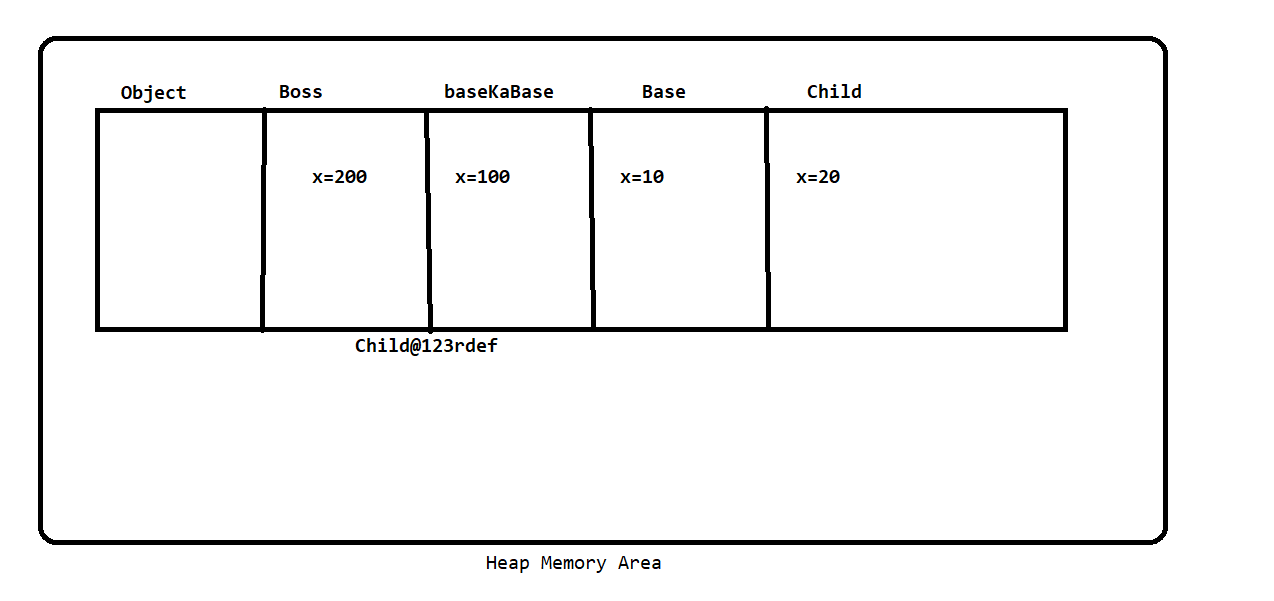
**About Type:-**

1. In java every primitive variable have it’s type.**int x=10;**
2. In java every object have their type. e.g new A();
3. In java every reference variable has their type. A a;

In java any child class reference is child type and parent type too. It means child class object (reference) are tread as the parent class instance.

**Question: -** In case of Inheritance when we are creating the object of Child class parent class object are created or not? If not then how parent class data member gets memory. If yes then how?

**Answer: -** No, Object of parent class is not created. All the data of parent get memory into the child class object as shown the diagram below.

no parent class object is not created. All the parent data gets memory into child class object. We can see in the below diagram: **

***\*\*\*Rule****: - We can call the parent class members with child reference but we can’t call the child class personal data with parent reference.(****V.V.Imp****)\*\*\**

***Instanceof: -*** The **java *instanceof* operator** is used to test whether the object is an instance of the specified type (class or subclass or interface). The *instanceof* in java is also known as type *comparison operator* because it compares the instance with type. It returns either true or false. If we apply the *instanceof* operator with any variable that has null value, it returns false.

//Example and Syntax to use instanceof operator  .

    class Sample1{

         public static void main(String args[]){

         Sample1 s=new Sample1();

         System.out.println(s instanceof  Sample1);  //true

         }

        }

package resubality;

class Boss

{

int x=200;

}

class BaseKaBase extends Boss

{

int x=100;

}

class Base extends BaseKaBase

{

    int x=10;

}

public class Child extends Base

{

    int x=20;

    void show(Base b1,BaseKaBase b2,Boss b3)//upcasting.....

    {

//      System.out.println(b1.x);

//      System.out.println(b2.x);

//      System.out.println(b3.x);

//      System.out.println(super.x);

//      System.out.println(x);

//

        System.out.println(((Base)this).x);

        System.out.println(((BaseKaBase)this).x);

        System.out.println(((Boss)this).x);

        System.out.println(x);

    }

    public static void main(String[] args)

    {

        Child c =new Child();

        //Base v=c;

        c.show(c,c,c);

        System.out.println(c instanceof Child);

        System.out.println(c instanceof BaseKaBase);

        System.out.println(c instanceof Boss);

        System.out.println(c instanceof Base);

  }

    }

**Method Overriding**

*@overriding*

When in a program the parent class method name and child class method name are same this concept is called method overriding .In other words we can say that when child class rewrites the parent class method with same name and same signature so we can say that child class overrides the parent class method.

//Example

class Parent {

    Void show()

    {

    }

    }

    class Child extends Parent{

    Void show()         // @overridden method

    {

    }

    }

**Rules:**

1. There must be a relation of parent and child between classes for method overriding.
2. The name of the method and its signature must be same.
3. In Case of method overriding the preference always goes to child class overridden method.
4. Static, final and private methods can’t be overridden.
5. If the method of parent class is static and child is non static then it’s not valid also it’s opposite is not valid. But if both methods are static then it’s valid. (This concept is called method hiding)
6. In Java there are 4 type of access rule
   1. Public
   2. protected
   3. default
   4. Private

**Note: This sequence is stronger sequence. But case of method overriding either same or weaker to stronger.**

**#Example**

package inheritance;

class Base {

    protected void show ()

    {

        System.out.println("this show message");

    }

}

public class Child extends Base {

    public void show ()

    {

        System.out.println("this is child message");

    }

    public static void main(String[] args) {

        Base s =  new Child();

        s.show();

    }

**\*\* From jdk 1.5 method overriding can also be done by changing the return type of the method, but the conditions below must be followed.**

**Case 1:** Both class methods must have return type of reference type

Primitive return type is not allowed.

\*The classes which is used as a return type they must have relation of parent and child.

**Case 2:** Parent Class method should return parent class reference and child class method should return child class reference

The Above Case is known as **COVARIANT RETURN.**

**COVARIANT RETURN:** Before java 1.5 it’s not possible to override a method by changing its return type but from java 1.5 we can override a method by changing return type. (But this must satisfy case 1 and 2).

IF the return type of parent and child class varies then this is known as covariant return.

**Uses of constructer in case of inheritance:**

In java before executing any child class constructer firstly execute its immediate parent class default constructor implicitly.

**Why?**

Ans: In order to initialize parent data member.

**But How?**

Ans: To call the Constructor of parent class in case of inheritance the Compiler implicitly added **super()** in the very beginning of the constructor of Child class. by using super we can call the dc of the immediate parent class.

package Inharitence;

class A

{

A()

{

    System.out.println("Defualt constructer A");

    }

}

class B extends A

{

B()

{

    System.out.println("Defualt constructer B");

    }

}

class C extends B

{

C()

{

    System.out.println("Defualt constructer C");

    }

}

public class Ineherite extends C {

    Ineherite()

    {

        System.out.println("Defualt constructer Ineherite");

        }

    public static void main(String[] args) {

        Ineherite s = new Ineherite();

    }

}

**Note:** we cannot use **this ()** and **super()** together in a constructor. Because call to this must be first statement in a constructor body and call to super must be first statement a constructor body. By default it is using **super().**

package Inharitence;

class A

{

A()

{

    super();

    System.out.println("Defualt constructer A");

    }

}

class B extends A

{

B()

{

    super();

    System.out.println("Defualt constructer B");

    }

}

class C extends B

{

C(int x)

{

    this();

    System.out.println(x);

    }

C()

{

    super();

    System.out.println("Defualt constructer C");

    }

}

public class Ineherite extends C {

    Ineherite()

    {

        super(40);

        System.out.println("Defualt constructer Ineherite");

        }

    public static void main(String[] args) {

        Ineherite s = new Ineherite();

    }

}

**Constructor Chaining:**

It is process of calling multiple constructors in a chain on single object.

package inharitence;

class A

{

    A(int a)

    {

        System.out.println(a);

    }

A()

{

    super();

    System.out.println("Defualt constructer A");

    }

}

class B extends A

{

    B(int x, int y)

    {

        super(65);

        System.out.println(x);

        System.out.println(y);

        }

    B(int x)

    {

        this(25,65);

        System.out.println(x);

        }

}

class C extends B

{

C(int x)

{

    this();

    System.out.println(x);

    }

C()

{

    super(25);

    System.out.println("Defualt constructer C");

    }

}

public class Ineherite extends C {

    Ineherite()

    {

        super(40);

        System.out.println("Defualt constructer Ineherite");

        }

    public static void main(String[] args) {

        Ineherite s = new Ineherite();

    }

}

What is binding?

When a member call getting their member definition is called binding.

**OR**

Connecting a member call with their member definition is called binding.

\*Member means data and member functions.

**There are two types of binding**

1. Compile time binding (earlier binding)

2. Runtime binding (late time/dynamic binding)

**Compile Time Binding:** When a member call getting their member definition at compile time this is called compile time binding. In case of compile time binding compiler does its binding on the basis of reference variable.

**Dynamic Binding:** When a member call getting their member definition at runtime this is called dynamic binding. Runtime binding is done by JVM on the basis of object type.

**\*\*\*Except non-static methods /virtual methods/instance methods all other entities are bound statically.**

**Static Binding is done in case of all the below entities:**

* Static Data Member
* Non-static data member
* Static methods
* Constructor
* Initializers
* Private methods
* Final methods
* Method call by a ***super*** keyword

**\*\*Note:** At the time of compilation assembly instructions are putted by the compiler in the byte-code in both types of binding.

**Instructions include:**

1. Invoke static: static binding to be done.
2. Invoke virtual: dynamic binding to be done.
3. Invoke special: this is also static binding but a special type of case.

class Parent{

    {

        System.out.println("Init of parent");

    }

    int x=100;

    static int s=200;

    Parent (){

        System.out.println("Dc of  parent  const...");

    }

    static void show() {

        System.out.println("Show method of Parent");

    }

    void display() {

        System.out.println("Display  method of Parent");

    }

}

public class Children  extends Parent{

int x=10;

static int s=20;

{

    System.out.println("init of child");

}

Children(){

    System.out.println("DC of Child ...");

}

void display() {

    System.out.println("Display  method of Child");

}

static void show() {

    System.out.println("Show method of Child");

}

public static void main(String[] args) {

    Parent c= new Children();

    System.out.println(c.x);

    System.out.println(c.s);

    c.show();

    c.display();

}

}

//cmd line argument dynamic binding

package suryaelectical;

class Parent{

    {

        System.out.println("Init of parent");

    }

    int x=100;

    static int s=200;

    Parent (){

        System.out.println("Dc of  parent  const...");

    }

    static void show() {

        System.out.println("Show method of Parent");

    }

    void display() {

        System.out.println("Display  method of Parent");

    }

}

public class Children  extends Parent{

int x=10;

static int s=20;

{

    System.out.println("init of child");

}

Children(){

    System.out.println("DC of Child ...");

}

void display() {

    System.out.println("Display  method of Child");

}

static void show() {

    System.out.println("Show method of Child");

}

public static void main(String[] args) {

    try {

        Class c =Class.forName(args[0]);

        Parent obj =(Parent)c.newInstance();

        System.out.println(obj.x);

        System.out.println(obj.s);

        obj.show();

        obj.display();

    } catch (Exception e) {

        e.printStackTrace();

    }

//  Parent c= new Children();

//  System.out.println(c.x);

//  System.out.println(c.s);

//  c.show();

//  c.display();

}

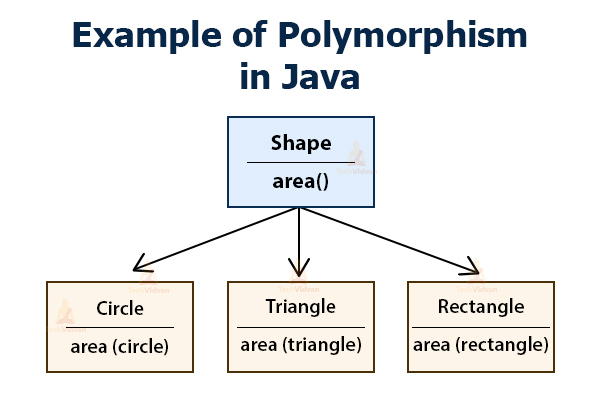
}

**POLYMORPHISM**

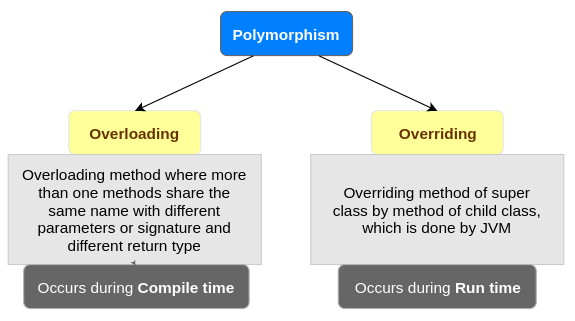
Polymorphism is made up of two words Poly and Morph

1. Poly means: Many
2. Morph means shapes.

Thus **Polymorphism** is the ability of an object to take on many forms. The most common use of **polymorphism** in OOP occurs when a parent class reference is used to refer to a child class object. Any Java object that can pass more than one IS-A test is considered to be **polymorphic**. See the image below …



**Polymorphism is of two types: -** Compile Time Polymorphism & Runtime Polymorphism



1. **Compile time polymorphism: -** Compile time polymorphism or static method dispatch is a process in which a call to an overloading method is resolved at compile time rather than at run time. In this process, we done overloading of methods which is called through the reference variable of a class here no need to superclass.

Method overloading: - If a class have multiple methods by same name but different parameters, it is known as Method Overloading.

There are two ways to overload the method in java: -

* By changing number of arguments

//Method overloading by changing number of arguments

class Calculation{

    void sum(int a,int b)

    {

       System.out.println(a+b);

    }

    void sum(int a,int b,int c){

       System.out.println(a+b+c);

     }

    public static void main(String args[]){

    Calculation obj=new Calculation();

    obj.sum(10,10,10);

    obj.sum(20,20);

    }

  }

OUTPUT:

30

40

* By changing the data type

//Method overloading by changing the parameters datatype

class Calculation{

    void sum(int a,int b){

      System.out.println(a+b);

     }

    void sum(double a,double b){

      System.out.println(a+b);

    }

    public static void main(String args[]){

    Calculation obj=new Calculation();

    obj.sum(11.5,11.5);

    obj.sum(30,20);

    }

  }

\*\* It can also be done by changing both number of parameters and data type of parameters as shown in the above code.

**Constructor overloading: -** Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

//Example of constructor overloading

class Demo{

    int  value1;

    int  value2;

    /\*Demo(){

     value1 = 10;

     value2 = 20;

     System.out.println("Inside 1st Constructor");

   }\*/

   Demo(int a){

    value1 = a;

    System.out.println("Inside 2nd Constructor");

  }

  Demo(int a,int b){

  value1 = a;

  value2 = b;

  System.out.println("Inside 3rd Constructor");

 }

 public void display(){

    System.out.println("Value1 === "+value1);

    System.out.println("Value2 === "+value2);

}

public static void main(String args[]){

  Demo d1 = new Demo();

  Demo d2 = new Demo(30);

  Demo d3 = new Demo(30,40);

  d1.display();

  d2.display();

  d3.display();

}

}

1. **Run time polymorphism: -** Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

#Example:

class Bank{

    float getRateOfInterest(){return 0;}

    }

    class SBI extends Bank{

    float getRateOfInterest(){return 8.4f;}

    }

    class ICICI extends Bank{

    float getRateOfInterest(){return 7.3f;}

    }

    class AXIS extends Bank{

    float getRateOfInterest(){return 9.7f;}

    }

    class TestPolymorphism{

    public static void main(String args[]){

    Bank b;

    b=new SBI();

    System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());

    b=new ICICI();

    System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());

    b=new AXIS();

    System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());

    }

    }

**OUTPUT:-**

**SBI Rate of Interest: 8.4**

**ICICI Rate of Interest: 7.3**

**AXIS Rate of Interest: 9.7**

#Example:

class Shape{

    void draw(){System.out.println("drawing...");}

    }

    class Rectangle extends Shape{

    void draw(){System.out.println("drawing rectangle...");}

    }

    class Circle extends Shape{

    void draw(){System.out.println("drawing circle...");}

    }

    class Triangle extends Shape{

    void draw(){System.out.println("drawing triangle...");}

    }

    class TestPolymorphism2{

    public static void main(String args[]){

    Shape s;

    s=new Rectangle();

    s.draw();

    s=new Circle();

    s.draw();

    s=new Triangle();

    s.draw();

    }

    }

//Master Remote polymorphism example

package plolymorphism;

public class MaterRemote {

    void channelChange() {}

    void colorChange() {}

    void volChange() {}

}

class OperateTv{

public static void main(String[] args) throws ClassNotFoundException, InstantiationException, IllegalAccessException {

    Class c=Class.forName(args[0]);

    MaterRemote m=(MaterRemote)c.newInstance();

    m.channelChange();

    m.colorChange();

    m.volChange();

}

}

class tv1 extends MaterRemote{

    void channelChange() {

        System.out.println("channel of tv1 changed");

    }

    void colorChange() {

        System.out.println("color ovf tv1");

    }

    void volChange() {System.out.println("Vol of tv1 change");}

}

class tv2 extends MaterRemote{

    void channelChange() {

        System.out.println("channel of tv2 changed");

    }

    void colorChange() {

        System.out.println("color ovf tv2");

    }

    void volChange() {System.out.println("Vol of tv2 change");}

}

class tv3 extends MaterRemote{

    void channelChange() {

        System.out.println("channel of tv3 changed");

    }

    void colorChange() {

        System.out.println("color ovf tv3");

    }

    void volChange() {System.out.println("Vol of tv3 change");}

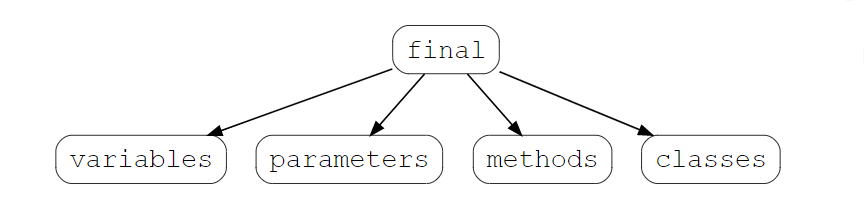
}

Final keyword in java

In Java, the final keyword can be used while declaring an entity. Using the final keyword means that the value can’t be modified in the future.

Final keyword can be used with:

1. class
2. Method
3. Data member
4. Local variable



## Final variables

If a variable is declared with the final keyword, its value cannot be changed once initialized. Note that the variable does not necessarily have to be initialized at the time of declaration. If it’s declared but not yet initialized, it’s called a **blank final variable**.

Blank final variables should be initialized in constructor or init block.

// declaring a final variable

class FinalVariable {

        final int var = 50;

        var = 60 //This line would give an error

}

## Final methods

A method, declared with the final keyword, cannot​ be overridden or hidden by subclasses.

// declaring a final method

class Base{

    public final void finalMethod(){

        System.out.print("Base");

    }

}

class Derived extends Base{

    public final void finalMethod() { //Overriding the final method throws an error

        System.out.print("Derived");

    }

}

## Final classes

A class declared as a final class, cannot be subclasses it means final class can’t be inherited.

// declaring a final class

final class FinalClass {

  //...

}

class Subclass extends FinalClass{ //attempting to subclass a final class throws an error

  //...

}

## Final parameters

If you ever see the final keyword with a parameter variable, it means that the value of this variable cannot be changed anywhere in the function.

class finalParameter {

    public static void example( final int parameter ) {

      parameter = 4; //attempting to reassign a value to a parameter throws an error

    }

}

Abstraction in Java

**In java abstract is a keyword which can only be used with:**

1. Methods
2. Classes

**Abstract Method:**

If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as an abstract.

1. The **abstract** keyword is used to declare the method as abstract.
2. You have to place the **abstract** keyword before the method name in the method declaration. E.g.,  *public void get();*
3. An abstract method contains a method signature, but no method body.
4. Instead of curly braces, an abstract method will have a semicolon (;) at the end.

Declaring a method as abstract has two consequences: -

1. The class containing it must be declared as abstract.
2. Any class inheriting the current class must either override the abstract method or declare itself as abstract.

**Abstract Class:**

A class which contains the abstract keyword in its declaration is known as an abstract class. E.g.,  *abstract class Employee{ }*

1. Abstract classes may or may not contain abstract methods, i.e., methods without a body. . E.g.,  *public void get();*
2. But, if a class has at least one abstract method, then the class must be declared abstract.
3. If a class is declared abstract, it cannot be instantiated means we can’t create object of that class.
4. To use an abstract class, you have to inherit it from another class, and also you need to define/override all the abstract method of that class in your class.

package abstracion;

// Abstract class Employee

abstract class Employee {

       private String name;

       private String address;

       private int number;

       double sal;

// abstract method ComputePay()

       public abstract double computePay(); //Abstract method

       // Remainder of class definition

    }

class Developer extends Employee{

    @Override

    public double computePay() {

        sal=49000.99;

        System.out.println("Salary of developer is "+sal);

        return sal;

    }

}

class Tester extends Employee{

    @Override

    public double computePay() {

        sal=49000.99;

        System.out.println("Salary of Tester "+sal);

        return sal;

    }

}

public class DispSal{

    public static void main(String[] args) {

        Employee dev;

        dev= new Developer();

        dev.computePay();

        dev = new Tester();

        dev.computePay();

    }

}

**Abstraction:-**

It is the process of showing the essential functionality of an object and hiding the unessential complexity.

Show functionality and hide complexity.

Show function and hide implementation.

We always achieving abstraction on function not on data.

It is the process of creating generic method.

Generic method are those method which have only declaration and it’s implantation are provided by different-2 users as per need.

**Advance:-**

To achieve runtime polymorphism.

**Implementing of abstraction:-**

* 1. **abstract class (0 to 100 %)- partially implantation of abstraction**
  2. **Interface ------(100 %)- fully implantation of abstraction.**

**abstract keyword can be applied on following things:-**

1. **Class**
2. **Method**

**Note:- abstract keyword can’t applied on data member and constructor.**

**Abstract class:-**

When we put the abstract keyword with class then class is called abstract class. we can’t create the object of abstract class.

**abstract class Test {**

**}**

Abstract class may or may not have abstract method.

**Rule:-**If any normal class extends the abstract class the normal class need to override all the abstract method of abstract class otherwise child class have to be abstract.

We can have reference variable of abstract class.

**abstract method:-**

Any method become abstract if we use the abstracts keyword with method.

abstract method do not specify a body.

We can not have abstract static ,final and private method.

Abstract class can have constructor

**Interface.**

It is the blue print of class.

It is another way to achieve the abstraction.

It provides the fully implementation of abstraction.

It is the combination public static final data member and public abstract method.

It is the contract between two parties programmer and programming language.

**Syntax:-**

interface Identifier

{}

Rule:-

1. If a class wants to use and interface then we have to use the **implements** keyword.
2. If a class implements an interface then class need to override all the method of interface with public access specifier otherwise child class need be abstract.
3. A class can implements more than one interface simultaneously and can inherit another class also. In this case firstly we have use the extends then implements.
4. One interface can extends another interface.
5. One interface can extends more than one interface and this is called multiple inheritance
6. We can’t create the object of interface but we can have reference variable of interface.
7. Interface can’t inherit a class.
8. Class can’t inherit interface.

**String Handling**

**String-** In general String is the sequence of character or combination of character.

String is the implementation of array.

String is the array of character data type.

int x[];

char c[]={‘j’,’a’,’v’,’a’};

‘A’,‘B’,‘C’. “ABC”

The process of performing operation over the String (length count, con cat, replace, substring etc.) is called String handing.

In java in order to performed string operation , java provides following classes-

1. java.lang.String
2. java.lang.StringBuffer
3. java.lang.StringBuilder
4. java.util.StringTokenizer

In java String is the represented by an object of String class.

**String class:-**

In java String is the immutable sequence (unchangeable) of character.

This class is internally public and final.

This class extends object class and implements Serializable , Comparable, Char Sequence interface .

Immutable means in the one object of String we can not perform operation like- insert, delete, update etc.

**How to create the object of String class:-**

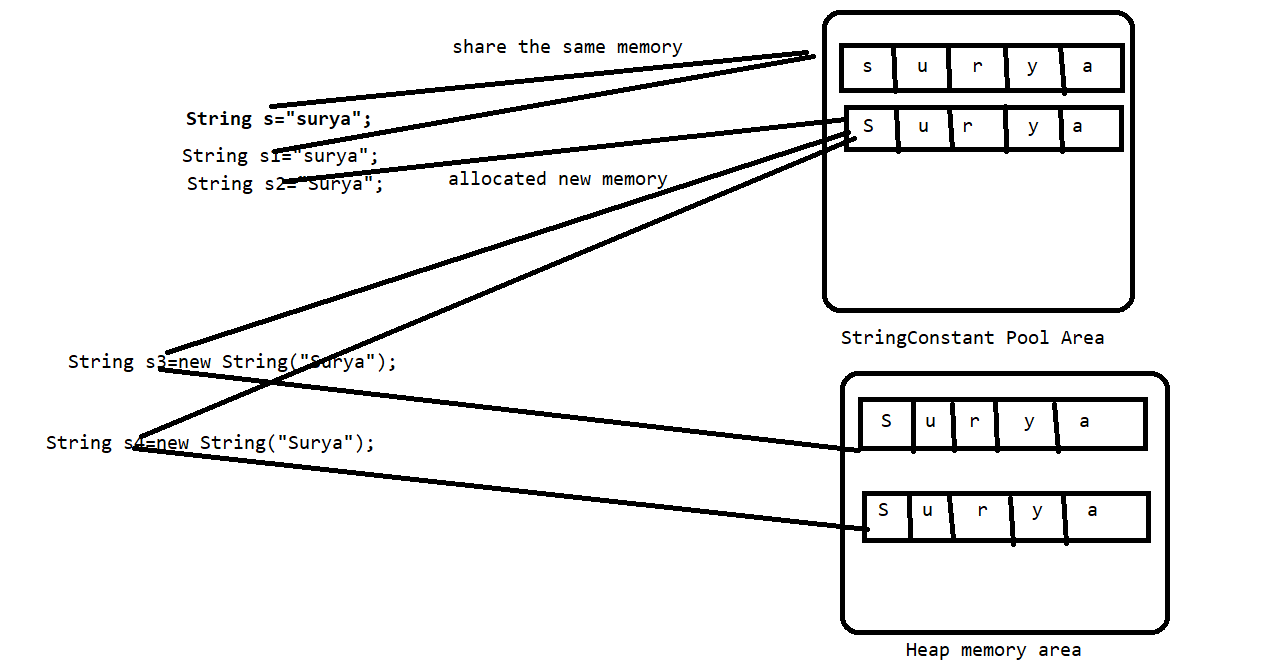
There are two ways ate used to create the object of String class.

1. without new (By String literal )------e.g **String s=”surya”;** “helllo”
2. with new e.g.  **String s=new String(“surya”);**

In java all the String literal are treated the String class object.

**Note:-** In java String , Wrapper classes , array objects can be created without new.

1. In case of without new we are creating the string object memory is allocated in String Constant pool area (part of method area). Pool areas does not allowed **duplicate** element. (It means whenever String object is created using this approach first it check the same name of string object is available or not if not then create a new memry otherwise share same memory location ). In this approach memory is optimized .
2. In case of new keyword memory is allocated in both area (heap as well as pool area). Here duplicate element are allowed.



**String Comparison:-**

1. By equals() and equalsIgnoreCase() method
2. By == operator
3. By compare To() and compareToIgnoreCase

length Vs length()

**package** stringhandling;

**public** **class** TestStr1

{

**public** **static** **void** main(String[] args)

{

String s="surya";

String s1="surya";

String s2="Surya";

System.***out***.println(s.equals(s2));

System.***out***.println(s.equalsIgnoreCase(s2));

System.***out***.println(s==s2);

System.***out***.println(s.equals(s1));

System.***out***.println(s==s1);

System.***out***.println(s.hashCode());

System.***out***.println(s1.hashCode());

TestStr1 t1=**new** TestStr1();

TestStr1 t3=t1;

TestStr1 t2=**new** TestStr1();

System.***out***.println(t1.equals(t2));

System.***out***.println(t1.equals(t3));

System.***out***.println(t1==t2);

String s3=**new** String("surya");

String s4=**new** String("surya");

System.***out***.println(s3.equals(s4));

System.***out***.println(s3==s4);

String s5=s3.intern();

System.***out***.println(s==s5);

}

}

**String Buffer:-**

A thread-safe, mutable sequence of characters. A string buffer is like a String but can be modified.

It means in the one object of String Buffer we can do modification like- insert update etc.

String Buffer class object can be created by new keyword only.

**Commonly used constructor of String Buffer:-**

|  |
| --- |
| [**StringBuffer**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\lang\java\lang\StringBuffer.html#StringBuffer--)()  Constructs a string buffer with no characters in it and an initial capacity of 16 characters. |
| [**StringBuffer**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\lang\java\lang\StringBuffer.html#StringBuffer-java.lang.CharSequence-)(**[CharSequence](file:///D:\\Users\\BKU\\SuryaPrakashTripathi\\Desktop\\doc\\src\\java\\lang\\java\\lang\\CharSequence.html" \o "interface in java.lang)** seq)  Constructs a string buffer that contains the same characters as the specified CharSequence. |
| [**StringBuffer**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\lang\java\lang\StringBuffer.html#StringBuffer-int-)(int capacity)  Constructs a string buffer with no characters in it and the specified initial capacity. |
| [**StringBuffer**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\lang\java\lang\StringBuffer.html#StringBuffer-java.lang.String-)([**String**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\lang\java\lang\String.html) str) |

In case of default constructor of String Buffer when we are creating the object then it’s default capacity is 16 characters. and when inserted more values beyond the size of array then this array internally automatically increased with bellow formula.

**Oldlength \* 2 + 2**

**Or**

**2(Oldlength+1)**

**StringBuilder:-**

A mutable sequence of characters. This class provides an API compatible with StringBuffer, but with no guarantee of synchronization.

It is not a thread safe class.

**Exception Handling :-**

There are five keywords are used in java to handle the exception:

**1.try**

**2.catch**

**3.finall**

**4.throw**

**5.throws**

All the above are keyword but try, catch & finally are called block and throw and throws are called clause.

**Exception:-**

It is the abnormal situation which occurs in the program at runtime due to this interrupts the normal flow of the program. And the rest of code will not be executed . In exception handling we learn how to handle this abnormal situation in the program.

E.g:-

Class A

{

Stamemt1….

Stamemt2….

Stamemt3….

Stamemt4….

Stamemt5….

}

Here suppose an exception is occur at statement no-3 so the rest statement will not executed but in exception handling we handle how to maintain this normal flow of the program.

**Error:-** there are three type of error-

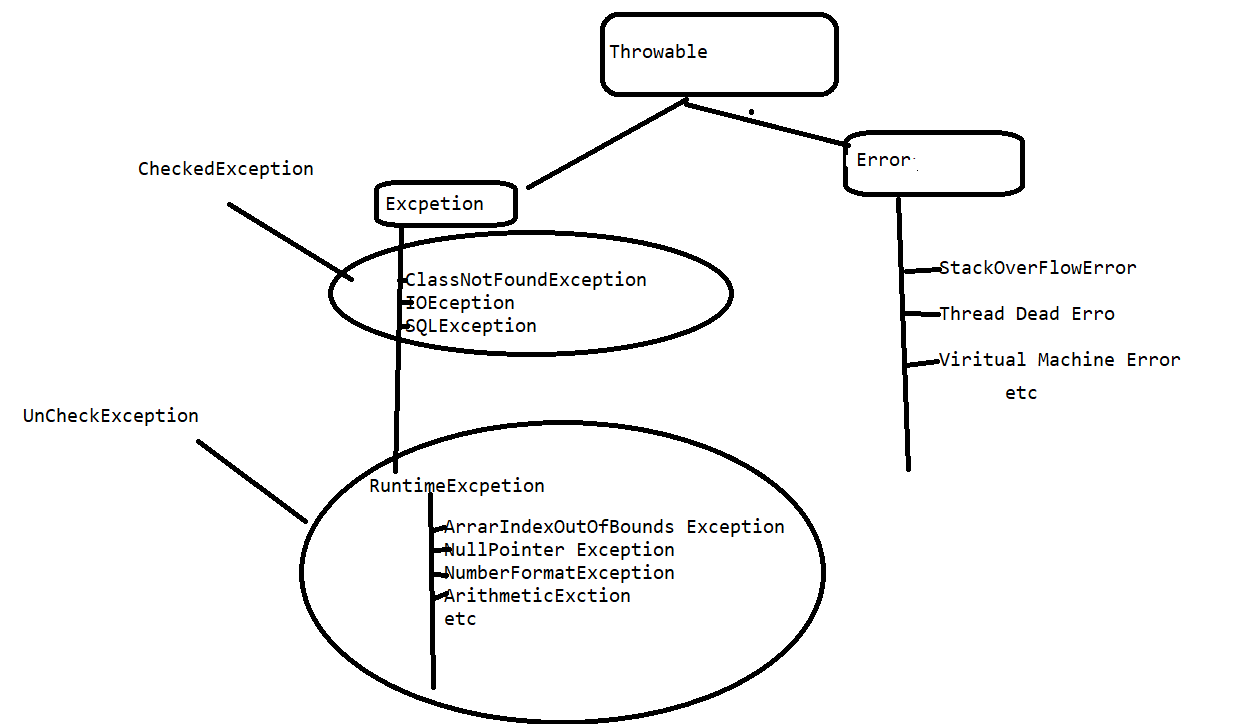
1. Syntax error
2. Logical error
3. Runtime error

Exception handling is the process of handling the runtime errors.

Exception can be handle but errors can’t be handle.

**From coding pint of view exception and error is an object is java and it is represented by a class called java. lang. Exception and java. lang. Error respectively .**

**In java java .lang package contains so many classes to represent exception and errors and they store in the hierarchical manner**



try

{

Java statement-.----------

}

Java block contains those set of statement where exception may occur.

try must be followed by either catch or finally.

**catch**(TypeofExcption refvariable )

{

}

Single try can have multiple catch block but at a time only one catch block is executed. Because in a single try only one exception occur at a time.

Between try and catch only spaces are allowed not any statement.

Comments are allowed.

If exception occur in try then it directly to go catch means if in this try block suppose we have 10 statement and exception occur at first statement then remaining line not reachable(9 statement) .

**What Happened When Exception Occur:-**

When exception occur in try block then firstly JVM matches the it’s corresponding class in the exception hierarchy if It found the matching class then

1. load the exception class.
2. Create the object of exception class
3. Throw the exception object into corresponding catch
4. If corresponding catch is found, then exception is handling and after executing all statement of catch block the normal flow of program are maintained.
5. If corresponding catch not found, then JVM put own handler and the normal flow of program has not maintained.

**Single Try With Multile Catch Block:-**

**try**

{

}

**catch(IllegalArgumentException e)**{

some code;

}

**catch(SecurityException e)**{

someCode();

}

**catch(IllegalAccessException e)**{

someCode();

}

**catch(NoSuchFieldException e)**{

someCode();

}

**Multiple Try Catch:-**

**try**

{

}

**catch(IllegalArgumentException e)**{

some code;

}

try{

**catch(SecurityException e)**{

someCode();

}

try

{

**catch(IllegalAccessException e)**{

someCode();

}

try{

**catch(NoSuchFieldException e)**{

someCode();

}

**Nested Try Catch:-**

* A try statement can be inside block of another try.
* Sometimes Condition may arise where part of block may cause one error and entire block may cause another error.So we can use in java, nested try.

**try**{

statement 1;

statement 2;

**try**{

statement 1;

statement 2;

}

**catch (Exception1 e)**{

//statements to handle the exception

}

}

**catch (Exception2 e2)**{

//statements to handle the exception

}

* When nested try blocks used, inner try block is executed first.
* Any Exception thrown in the inner try block is caught the corresponding catch block.
* If matching catch block is not found So catch block of outer block are examining until all nested try statement exhausted.
* If no matching blocks are found,So java Runtime Ennvironment handles the execution.

**Finally:-** It is always executing block. Which execute every time either exception occur or not.

1. try

{

}

finally

{

}

2.try

{

}

catch()

{

}

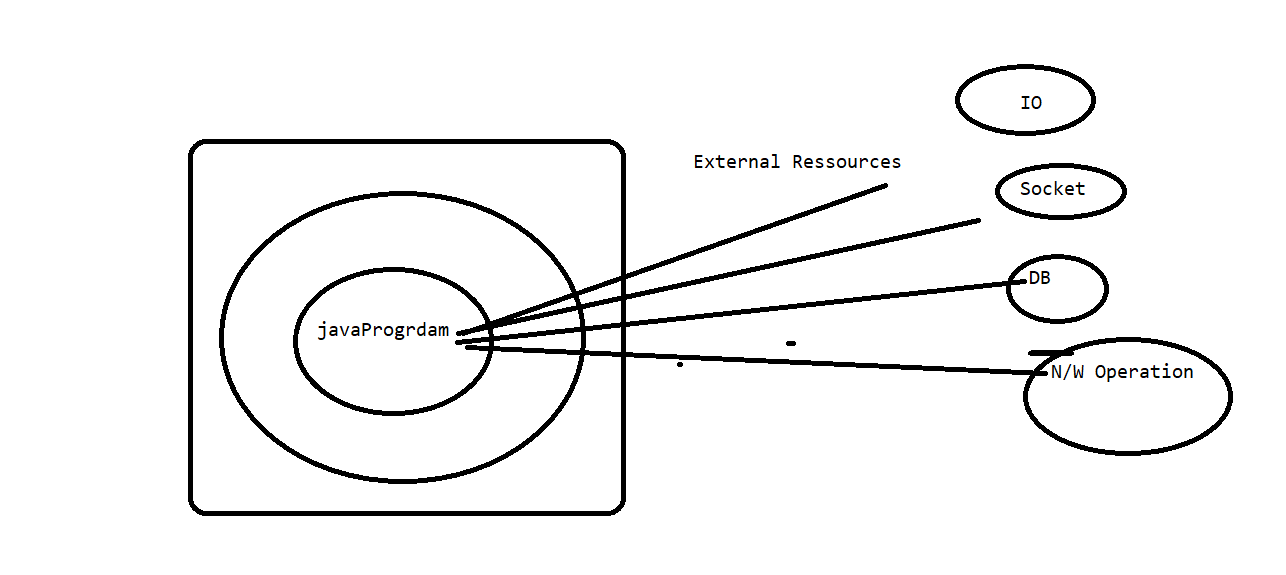
finally

{

}

The main purpose of finally block is to write cleaup code in the program.

* 1. If exception not occurred then finally executed and rest of code will executed
  2. If exceptions occurred but not handle then finally executed. And rest code will not executed.
  3. If exception occurred and handle the finally executed and rest of code will also be executed.



Is there any situation in which finally will not be execute.

When we explicitly terminating the program.

System.exit(0);

Can a return statement of method stop the finally.

**Checked Vs. Uncheck Exception :-**

**Partial Checked Vs. Uncheck Exception :-**

Checked exception are known by compiler at compile time and uncheck exception are known by JVM at runtime.

**Throws:**- This keyword is used to provide the indication to the end the what type of exception may occur in the program.

public void show() throws Type of Exception

{

}

**Throw:-** This keyword is used to throwing an exception explicitly in the program.

The main purpose of throw keyword is used to throw an object of custom exception.

Throw new Type Of Exception ();

**Exception Propagation:-**

If a method have throws with it. Then we must have to follow the below rule:-

1. If we are calling a method which have throws some exception then into the called method we need to handle same exception in try and catch otherwise throws same exception with method.

Throw is used to throwing an exception explicitly where as throws is a part of method signature and it is used to provide the indication to the end use that what type of exception may occur in the program.

Throw is the part of method definition and throws is the part of method signature.

We can throws multiple exception at a time separated by comma but we can throw only one exception at a time.

We can throw any exception but generally throws are used the in case of checked exception.

**Use of Throws In case of method overriding and Constructor creation :-**

1. **If parent class method does not throwing an exception.**
   * 1. Child class overridden method can throw only unchecked exception not check exception.
2. If parent class method throwing an exception.
   1. Child class method does not throw an exception.
   2. Child class method can throws same exception.
   3. Child class method can throws child exception but **can not throws parent exception**
3. In case of constructor we need to throws same exception and parent exception in child class constructor.

**Custom Exception:-**

When we are creating own exception class as per requirement then it is called custom exception . when ever we are crating custom exception in this case we need extends some predefine exception class in our own class.

**An Exception is said to be fully checked if and only if all the child classes are also checked exception, otherwise it is called partially checked exception.**

**Example:**

**IOException:- fully checked exception**

**Exception:- partially checked exception, as it's subclasses include both checked and unchecked Exceptions.**

**Throwable:- partially checked exception, as it's subclasses include both checked and unchecked Exceptions.**

**Collection Framework:-**

1. It is used to achieve the Data Structure functionality in java.
2. Data Structure is the concept which facilities us to store the in organized manger.
3. It was introduced in JDK1.2.
4. Before jdk1.2 we also achieve the data structure functionality, but they are not a framework now they are called legacy collection.
5. Collection represent group of object known as element, the object could be homogenous or heterogeneous.
6. Why collection – so that we can further perform some operation on that – maximum, short , mail.
7. Difference between collection(array list) and array – array is collection of homogenous type of data but collection have homogenous as well as heterogeneous. **Once we are creating the array it’s size became fixed and we can not dynamically increase and decrease the size of array where as in collection framework it is possible**. We can crate the array by both ways without new and with new but in case collection we are only using new keyword.

int []=new int[10];

Student []=new Student[10];

Object []=new Object[10000];

**API**:- set of inbuilt libraries or predefined functionality in any language. Whose from is varies from language to language. In c and C++ it is in the form header file and in java it is in the form of classes interfaces and package.

**Framework** :- It is set of API which work on the predefined algorithm and targeted to specific domain. Framework made the task easy.

**Framework Specific domains.**

Collection---------------------------------------------------D.S

JDBC----------------------------------------------------------Database connectivity

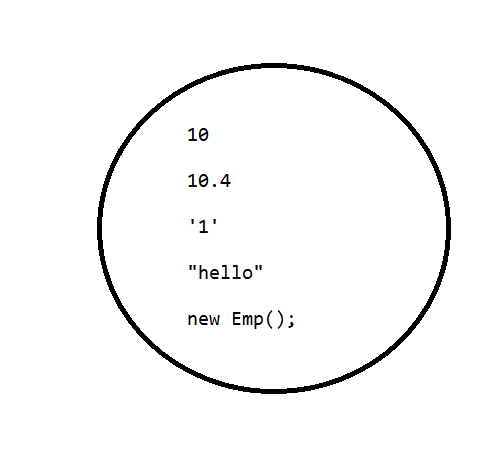
JEE------------------------------------------------------------WebApp & Enterprise App

Struts----------------------------------------------------------MVC

Hibernate……………………………………………………………..ORM

Spring……………………………………………………………………AOP & IOC etc.

**Advantage**:- It is use the simplify the development and maintenance of application software. Framework never wrong output because it is already tested.

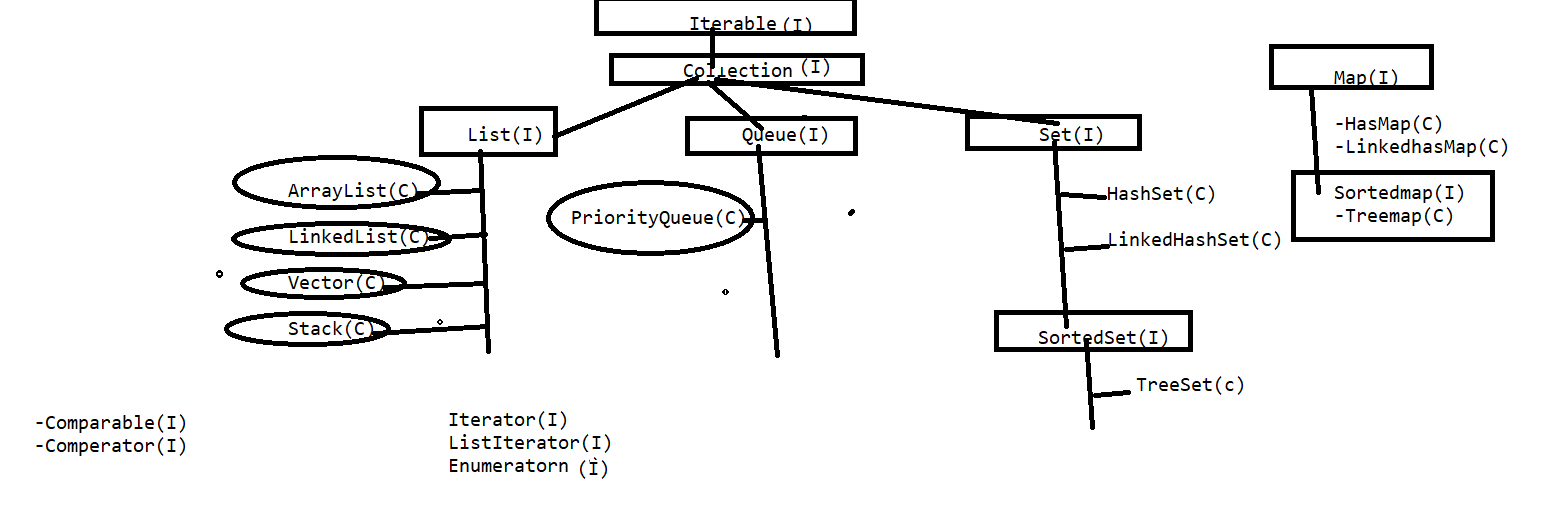


All the collection framework API are found in **java. util.** package.

Depending on behaviour collection API’s are categorised in so may parts.

1. **List**
2. **Set**
3. **Queue**
4. **Map**

**From coding point of view collection is the interface and collections is the class.**



Methods of Collection Interface :-

public abstract int size();

public abstract boolean isEmpty();

public abstract boolean contains(java.lang.Object);

public abstract java.util.Iterator<E> iterator();

public abstract java.lang.Object[] toArray();

public abstract <T> T[] toArray(T[]);

public abstract boolean add(E);

public abstract boolean remove(java.lang.Object);

public abstract boolean containsAll(java.util.Collection<?>);

public abstract boolean addAll(java.util.Collection<? extends E>);

public abstract boolean removeAll(java.util.Collection<?>);

public boolean removeIf(java.util.function.Predicate<? super E>);

public abstract boolean retainAll(java.util.Collection<?>);

public abstract void clear();

public abstract boolean equals(java.lang.Object);

public abstract int hashCode();

public java.util.Spliterator<E> spliterator();

public java.util.stream.Stream<E> stream();

public java.util.stream.Stream<E> parallelStream();

**Collection Interface:-**

The root interface in the *collection hierarchy*. A collection represents a group of objects, known as its *elements*. Some collections allow duplicate elements and others do not.

**List interface**:-

It is the child interface of collection which does allowed the duplicate elements and have also features of index.

And also maintain the insertion order of elements.

**Implementation(concrete classes ) of List interface:-**

1. Array List
2. LinkedList
3. Vector
4. Stack

**Array List:-** It is the concreate class of List interface , an object of this class is responsible to create a dynamic resizable array in memory with initial capacity is 10.

Constructor of array list:-

|  |
| --- |
| [**ArrayList**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\ArrayList.html#ArrayList--)()  Constructs an empty list with an initial capacity of ten. |
| [**ArrayList**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\ArrayList.html#ArrayList-java.util.Collection-)([**Collection**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\Collection.html)<? extends [**E**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\ArrayList.html)> c)  Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator. |
| [**ArrayList**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\ArrayList.html#ArrayList-int-)(int initialCapacity)  Constructs an empty list with the specified initial capacity. |

As we know that the initial capacity of arraylist is 10 when we added some more values beyond the size of array so default it increase the 50 % of actual size.

**How to Traverse the Element of Collection:-**

1. For loop
2. For Each
3. Iterator
4. List Iterator

**v.v.vImp**

**Iterator**:- It is an interface whose implementation are provided by different-2 collection as per requirement. An implementing class object of this interface is responsible to traverse the element in forward direction only.

**How to get the object of Iterator:-**

Iterator i=arralist.iterator();

Methods of Iterator:-

1. public Object next();
2. public boolean hasNext();
3. public void remove()

**How Iterator works internally**

Iterator internally uses the concept of iterator pointer to traverse the elements.

By default the pointer position is just first index(-Ve) . next method iterator is use to traverse the iterator pointer to next Index and return same value where iterator pointer is available. hash Next() method is used to check the collection data or not of yes the return true otherwise return false.

**List Iterator:-**

It is the Child interface of Iterator whose implementation are only provide by List interface . And It’s implementing class object of this interface is responsible to traverse the elements in both direction (forward and backword).

**Vector:-**

It is a also a implementing class of List whose one object is responsible to crate a dynamic resizable array. This is legacy class and synchronized also.

|  |
| --- |
| [**Vector**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\Vector.html#Vector--)()  Constructs an empty vector so that its internal data array has size 10 and its standard capacity increment is zero. |
| [**Vector**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\Vector.html#Vector-java.util.Collection-)([**Collection**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\Collection.html)<? extends [**E**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\Vector.html)> c)  Constructs a vector containing the elements of the specified collection, in the order they are returned by the collection's iterator. |
| [**Vector**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\Vector.html#Vector-int-)(int initialCapacity)  Constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero. |
| [**Vector**](file:///D:\Users\BKU\SuryaPrakashTripathi\Desktop\doc\src\java\util\java\util\Vector.html#Vector-int-int-)(int initialCapacity, int capacityIncrement)  Constructs an empty vector with the specified initial capacity and capacity increment. |

**Difference between Array List and Vector: -**

1. vector is a legacy class which was available science beginning of java but array list was introduce int jdk1.2 as part of collection framework.
2. Vector is synchronised and array list is not synchronized.
3. We can have synchronized as Array List but can not have unsynchronized as Vector **Note:- Synchronized methods are access sequentially in multithreading environment but non Synchronized method can be access simultaneously in multithreading environment , it means vector is thread safe and array list not. But vector is slower than array list. In Collections class we have a method called synchronised List by which we can make array List as synchronised .**
4. Once we added more value beyond the size of array list then array list recreated and by default it increases the 50 % of actual size. But in case of vector it increase the double of actual size.
5. We can provide the incremental factor in vector but not in array list.
6. Vector can use iterator, list iterator and enumeration to traverse the element but Array List can only use iterator, list iterator.

**Linked List:-** It is the concreate class of list interface an object of this class is responsible to creating doubly linked list in memory apart from doubly linked list this class also provide the implementation of Queue and Stack.

**What is Concurrent Modification ?**  
  
When one or more thread is iterating over the collection, in between, one thread changes the structure of the collection (either adding the element to the collection or by deleting the element in the collection or by updating the value at particular position in the collection) is known as Concurrent Modification  
  
**Difference between Fail Fast iterator and Fail Safe iterator**  
  
**Fail fast Iterator**  
  
Fail fast iterator while iterating through the collection , instantly throws Concurrent Modification Exception if there is structural modification  of the collection . Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Fail-fast iterator can throw ConcurrentModificationException in two scenarios :-

*Single Threaded Environment*  
  
After the creation of the iterator , structure is modified at any time by any method other than iterator's own remove method.  
   
*Multiple Threaded Environment*  
  
 If one thread is modifying the structure of the collection while other thread is iterating over it .

According to  [Oracle docs](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) , **the fail-fast behavior of an iterator cannot be guaranteed** as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness:**the fail-fast behavior of iterators should be used only to detect bugs.**  
  
  
**Interviewer : How  Fail  Fast Iterator  come to know that the internal structure is modified ?**  
Iterator read internal data structure (object array) directly . The internal data structure(i.e object array) should not be modified while iterating through the collection. To ensure this it maintains an internal  flag *"mods" .*Iterator checks the *"mods" flag* whenever it gets the next value (using hasNext() method and next() method). Value of *mods* flag changes whenever there is an structural modification. Thus indicating iterator to throw ConcurrentModificationException.

**Fail Safe Iterator :**  
  
Fail Safe Iterator makes copy of the internal data structure (object array) and iterates over the copied data structure.Any structural modification done to the iterator affects the copied data structure.  So , original data structure remains  structurally unchanged .Hence , no ConcurrentModificationException throws by the fail safe iterator.  
  
Two  issues associated with Fail Safe Iterator are :  
  
1. Overhead of maintaining the copied data structure i.e memory.  
  
2.  Fail safe iterator does not guarantee that the data being read is the data currently in the original data structure.  
  
According to [Oracle docs](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) , fail safe iterator is ordinarily too costly, but may be more efficient than alternatives when traversal operations vastly outnumber mutations, and is useful when you cannot or don’t want to synchronize traversals, yet need to preclude interference among concurrent threads. The "snapshot" style iterator method uses a reference to the state of the array at the point that the iterator was created. This **array never changes during the lifetime of the iterator, so interference is impossible and the iterator is guaranteed not to throw ConcurrentModificationException**.The iterator will not reflect additions, removals, or changes to the list since the iterator was created. Element-changing operations on iterators themselves (remove(), set(), and add()) are not supported. These methods throw UnsupportedOperationException.

# **5 Difference Between ArrayList And LinkedList In Java With Example**

Ad by Valueimpression

One of the start up [java interview questions](https://javahungry.blogspot.co.uk/2013/06/top-25-most-frequently-asked-core-java.html) on Collections topic is difference between ArrayList and LinkedList , interviewer may also ask  to write examples . We already discussed some other  basic interview questions like [difference between array and arraylist](https://javahungry.blogspot.com/2015/03/difference-between-array-and-arraylist-in-java-example.html) , [difference between arraylist and vector](https://javahungry.blogspot.ca/2013/12/difference-between-arraylist-and-vector-in-java-collection-interview-question.html) . In this post difference between arraylist and linkedlist , apart from the differences , we will also discuss the similarities , examples and when to prefer arraylist over linkedlist.  
  
**Difference between ArrayList and LinkedList in Java**  
**1. Implementation :**ArrayList is the resizable array implementation of list interface , while LinkedList is the Doubly-linked list implementation of the list interface.  
  
  
**2. Performance  :**Performance of ArrayList and LinkedList depends on the type of operation  
  
a. get(int index) or search operation :  ArrayList get(int index) operation runs in constant time i.e O(1)  while LinkedList get(int index) operation run time is O(n) .  
  
The reason behind ArrayList being faster than LinkedList is that ArrayList uses index based system for its elements as it internally uses array data structure , on the other hand ,  
LinkedList does not provide index based access for its elements as it iterates either from the beginning or end (whichever is closer) to retrieve the node at the specified element index.  
  
b. insert() or add(Object) operation :  Insertions in LinkedList are generally fast as compare to ArrayList.  
  
In LinkedList adding or insertion is O(1) operation . While in ArrayList, if array is full i.e worst case,  there is extra cost of  resizing array and copying elements to the new array , which makes runtime of add operation in ArrayList O(n) , otherwise it is O(1) .  
  
c. remove(int) operation :  Remove operation in LinkedList is generally same as ArrayList i.e. O(n).  
In LinkedList , there are two overloaded remove methods. one is remove() without any parameter which removes the head of the list and runs in constant time O(1) .  
The other overloaded remove method in LinkedList is remove(int) or remove(Object) which removes the Object or int passed as parameter . This method traverses the LinkedList until it found the Object and unlink it from the original list . Hence this method run time is O(n).  
  
While in ArrayList remove(int) method involves copying elements from old array to new updated array , hence its run time is O(n).  
  
**3.  Reverse  Iterator :**LinkedList can be iterated in reverse direction using descendingIterator() while there is no descendingIterator() in ArrayList , so we need to write our own code to iterate over the ArrayList in reverse direction.  
  
**4. Initial Capacity :**If the constructor  is not overloaded , then ArrayList creates an empty list of initial capacity 10 , while LinkedList  only constructs the empty list without any initial capacity.  
  
**5. Memory Overhead :**Memory overhead in LinkedList is more as compared to ArrayList as node in LinkedList needs to maintain the addresses of next and previous node. While in ArrayList  each index only holds the actual object(data).

**When to Use ArrayList and LinkedList :**  
In real world applications , you will more frequently use ArrayList than LinkedList. But in a very specific situations LinkedList can be preferred.  
1. ArrayList is preferred when there are more get(int) or search operations need to be performed as every search operation runtime is O(1).  
  
2. If application requires more insert(int) , delete(int) operations then the get(int) operations then LinkedList is preferred as they do not need to maintain back and forth like arraylist  to preserve continues indices.

Legacy Classes & Interfaces:-

1. Vector (C)
2. Stack (C)
3. Hash Table (C)
4. Enumeration (C)

**Lambda Expression:-**

**Set interface:-**

It is the interface which does not have any extra method it only changes the behaviour of add method in such a way that it does not allowed duplicate element. And not maintaining order as well.

**Implementation of Set interface: -**

1. Hash Set
2. Linked Hash Set

**Map Interface**:-

It is the part of collection framework but not a collection(because of it does not extends collection interface).

It store the data in the form of key and value pair.

Internally map interface uses the Entry interrace

hashMap

Linkedm

JDBC (Java Database Connectivity)

It is the API by which we can connect a java program with data base.

All JDBC API’s are available in **java.sql** package.

Any database provides following ways to store the data-

1. CUI
2. GUI
3. API

There are furthers following ways are used to work with API-

1. By using vender specific API
2. By using ODBC API
3. By using JDBC API

ODBC is the combination of two things:-

1. ODBC API
2. ODBC Driver

ODBC API is the declaration of methods in c language.

ODBC driver is the implementation of ODBC API

JDBC is the combination of two things: -

1. JDBC API
2. IDBC Driver

JDBC API is the declaration of methods in java language.

JDBC driver is the implementation of JDBC API

1. Type 1- JDBC-ODBC Bridge
2. Type-2 Native API
3. Type-3 Driver Server / Network Driver
4. Type-4 Driver or Universal / Thin Driver

Commonly used JDBC classes & Interface (JDBC API):-

1. Driver(I)
2. DriverManager(C)
3. Stataement(I)
4. PreparedStatement(I)
5. ResultSet(I)
6. Callable(I)
7. Connnection(I)
8. SQLException(C)
9. ResultSetMeteData

DatabaseMeta Data

Etc.

**Steps to connect a java program with any database:-**

1. Register the Driver With Driver manager in your java program.

a)- Call the static method registerDriver of Drivermanager.

e.g. DriverManager.registerDriver(new com.mysql.cj.jdbc.Driver());

or

by loading the diver class dynamically

Class c=Class.forName(“com.mysql.cj.jdbc.Driver”);

1. Get the Connection

Conncetion c=DriverManager.getConncetion(“url”,”usename”,”userpass”);

1. Get the required Statement.
2. Execute the Query
3. Close the connection

**Uses of Driver Manager:-**

1. Register the Drivers.
2. De-register the Drivers.
3. Stablish a connection between java application and data base.
4. Returns the created connection.

**Task-**

**1.Single Task**

It is the process of executing only one task at a time.

**2.Multitasking-**

It is the process of executing multiple task simultaneously at a time.

Multitasking is the just a concept its implementation is provided by –

1.Process

2.Thread.

MultiProcessing.

MutiThreading.