

JAVA CLASS: ->

02-Oct-2023: -

(Writes once run anywhere #WORA) Dialogue

Java is a versatile high level platform independent programming language which support oops concept very strongly.

3 categories of programming language: -

interpreted, compile, interpreted compile prog. Lang.

Java is interpreted compile programming language.

Java applications :->

- Console based application.
- Windows base application.
- Web base app.
- Enterprise base app.
- Distributed app.
- Mobile app.
- Embedded app.

Features of Java:->

1. Robust
2. Dynamic
3. Portable
4. Secure
5. Platform Independent(*important*)
6. Supports OOPS. (*important*)

7. Concurrency.
8. Distributed.
9. Flexibility.
10. Open Source.

JDK vs JRE vs JVM :->

JDK: - Java Development Kit.

JRE: - Java Runtime Environment.

JVM: - Java Virtual Machine.

JVM is platform dependent.

Java Strength:->

Platform Independency.

Platform => Operating System + Hardware Configuration.

Question: How it is platform independent?

Ans: -

because of byte coded class file and JVM JAVA is platform independent.

Compilation of java is **byte coded** then **source code** in **class** such as A. class.

Class file is Byte coded machine language, and converts into binary code by using JVM and then its compiled.

JVM is Interpreter of Java,
hence it is platform independent.

For every Operating system JVM is different.

2. OOPS:->

(Object Oriented Programming Structure)

four pillars of OOPS

Abstraction, Inheritance, polymorphism, Encapsulation.

ABSTRACTION :->

It is responsible to hide internal complexity and represent it in simplified manner.

E.g.: Container having book, pencil, pen, etc which are totally independent.

POLYMORPHISM :->

When we use one thing in different manner according to our requirement is called polymorphism.

ENCAPSULATION :->

It is responsible to hide internal complexity and binds them together for accomplishing some fruitful work.

E.g.: Car, in which without any single part car will not work.

INHERITANCE :->

If we establish parent child relationship between two or more than two entities, you authorize child entity to access any property of parent entity except private property.

It enhances the concept of Reusability of code.

DATA TYPES: -

It is the reserve keywords and identifiers which is responsible to define nature of data which it holds.

It is of two types: -

1. Primitive

- a. byte (1 bit), short (2 bit), int (4bit), long (8 bit) (default value = 0)
- b. float (4 bit), double (8bit) (default value = 0.0)
- c. char (2 bit) (default value = null)
- d. boolean (1 bit) (t/f) (default value = true)

2. Non primitive

a. Predefine

All class, interface, arrays, enumeration and object which is already present in system.

e.g., string

b. User defines

which is define by user.

Java has total 49 working keywords and always written in small letters

CLASS: -

It is collection of variables, methods and object.

It is collection of similar types of objects.

It defines object conceptually.

It is the blueprint of object. (Most important)

Syntax of class: -

class is keyword and always written with small alphabet

```
[Access specifiers] [Modifier] class ClassName {  
    // [] are optional
```

Access specifiers: -

It is the keyword which is responsible to define the scope of accessibility of any class, variable or method.

- a. public
- b. protected
- c. default
- d. private

Modifier: -

It is the reserved keyword which is responsible to modify the normal behaviour of any class, method or variable and assign certain restriction over that.

- a. final
- b. abstract
- c. static

ClassName: -

- a) ClassName can be any name upto 120 characters.
- b) ClassName should not start with numeric value.
- c) Only two symbols can be used \$ and _.
- d) ClassName should be start with capital letter.

Along with outer class we can use only use public and default access specifiers but along with inner class we can use all 4-access specifier.

Variables: -

- 1. Instance (Object) Variable: -** Any non-static variable which is define or declare inside the class but outside of any method becomes instance variable.

It has two properties: -

- a) It can be accessible anywhere and at any time inside the class.
- b) All the objects of the class have their own copy of instance variable any change made by any object doesn't affect any object.

- 2. Static (class) Variable: -** If we use static modifier along with any instance variable then that variable becomes static.

Properties: -

- a) It is same as instance variable
- b) All the objects of class share same static variable now any change made by any object affects the other object also

3. Local Variable: - Any variable define inside the particular method becomes local variable of that method.

It cannot be declared it is defined.

E.g., int a; declaration

 a = 10; initialization

 int a = 10; define / implementation

Properties: -

- a) Local variable cannot accessible outside the method.

Method: -

[Access specifiers] [Modifier] returnType methodName ([parameter]) {
 } // [] are optional

methodName should be start with small letter.

returnType: - it may be any data type whether it is primitive or non-primitive.

LOOPS: -

Use to access statement repeatedly.

Types of the loops: -

1. Do while: -

It accesses at least one time. When you want to handle cyclic iteration.

2. While: -

It is used to handle infinite iteration or we don't know how many iterations are there.

3. For: -

When no of iteration is finite i.e., know upper and lower value.

4. For each: -

It is use to access existing collection.

```
E.g., int arr [] = {1, 2, 3, 4, 5}
    for (int x: arr) {
        sop(x);
    }
```

ARRAY: -

Java. Utility package has all function regarding array.

Array is the collection of similar data having same data type.

Syntax: -

int arr []; → declaration.

arr = new int [5]; → initialization

int b [] = {1, 2, 3, 4, 5}; → implementation

Types of arrays: -

1. One dimensional.

2. Multi-dimensional.

3. Jagged Array: -

It is also type of 2d array. Elements in row are not same.

E.g.,

1	2	3	4	5
1	2	3		
4	5	6	7	

```
int ja [] [] = new int [3] [];
```

```
ja [0] = new int [5];
```

```
ja [1] = new int [3];
```

```
ja [2] = new int [4];
```

OR

```
{
```

```
    {1, 2, 3, 4, 5},
```

```
{1, 2, 3},  
{4, 5, 6, 7}  
}
```

By Default, java. lang is imported.

Mostly used package is java. util.

Object class is only package which is inherited.

METHOD OVERLOADING: -

It is also known as **static polymorphism**.

When we create more than one method having same name but differ in signature within same class is called method overloading.

Why we use method overloading?

Ans. Same functionality but different behaviour.

Signature = No of parameter + their data type.

E.g.,

```
void show () {...}  
void show (int x) {...}  
int show (double x) {...}  
void show (int x, double y) {...}
```

CONSTRUCTOR: -

It looks like a method but has certain differences over method: -

1. Its name must be same as class name.
2. It has no return type not even void.

Similarities with method: -

1. It can be parameterized like method that's why it can be overloaded like method.

```
Class Cdac {  
    Cdac (int a) {}
```



```
Cdac () {}  
}
```

Uses: -

1. It is useful to initialise the instance variable.
2. It is useful to allocate space in memory to all non-static properties of class.
3. It is useful to call other constructor of same class.
4. It is useful to call parent/base class constructor.

INHERITANCE: -

It is one of the properties of OOPS in which we establish parent child relationship between two or more than two classes and authorized child class (derived class) it accesses any property of parent class (base class) except private property by using **extends keyword**.

Types of inheritance: -

1. Single/Simple: -



E.g.,

package inheritance. single;

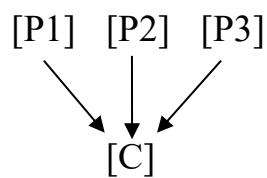
```
class Parent {  
    int a = 10;  
    void show () {  
        Sop ("a = "+a);  
    }  
}  
  
class Child extends Parent {  
    int b = 20;  
    void display () {  
        Sop ("b = "+b);  
    }  
}
```

```

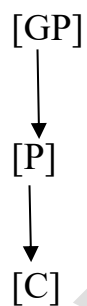
    }
}
public class SingleInheritance {
    public static void main (String [] args) {
        Child c = new Child ();
        c.show ();
        c.display ();
    }
}

```

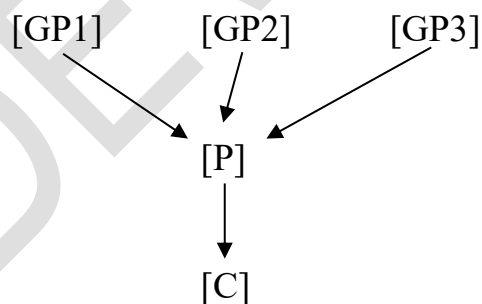
2. Multiple: -



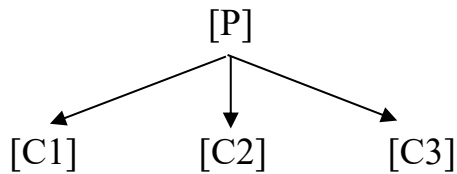
3. Multilevel: -



4. Hybrid: -



5. Hierarchical: -



Java does not support Multiple inheritance due to which it also not supports Hybrid inheritance.

METHOD OVERRIDING: -

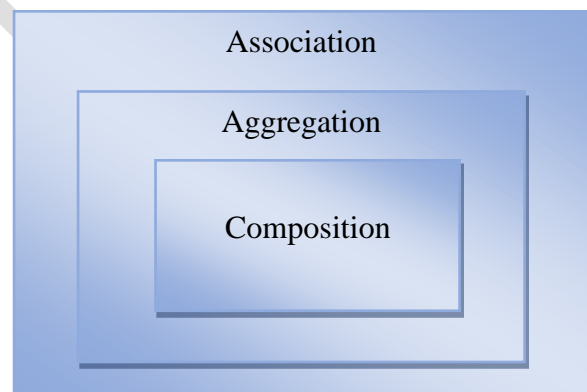
When we create a method in derived class which is already available in its base class i.e., the method having **same name and same signature** is called method overriding.

It is a type of run/dynamic polymorphism.

Has – a RELATION: -

Types of has – a relation: -

- Association
- Aggregation
- Composition



Association: -

If two different entity works together by using has -a relationship then it will follow the concept of association if both the entity can exist independently.

E.g., Person and address.

Aggregation: -

If group of one entity associate with other entity and if part of group survives independently then it is known as aggression relationship.

E.g., Library and book.

Composition: -

If both the entity which are involved in association cannot exist independently without each other then it would be the composition type relationship.

E.g., Car and engine.

THIS AND SUPER: -


this: -

It is a keyword which is responsible to refer class property it can be use to overcome variable hiding which is generated due to same local as well as instance variable.

Its second use is to call other constructor of same class. (It must be first line)

E.g.,

```
Class A {  
    int a = 10;  
    void show () {  
        int a = 100;  
        sop (a);  
        sop (this. a);  
    }  
}
```



super: -

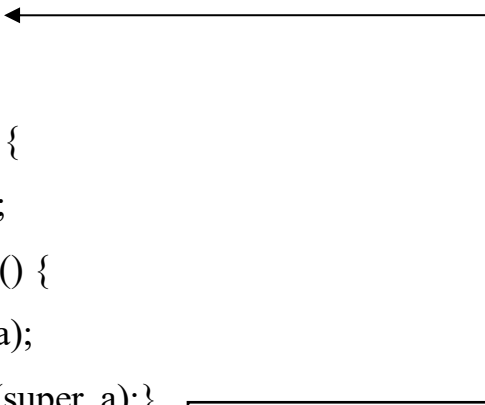
It is keyword which is responsible to overcome variable hiding which is generated due to same instance variable of derived class and base class also.

It is useful to call base class constructor.

It is also useful to call base class overridden method.

E.g.,

```
class P {  
    int a = 10; ←  
}  
class Q extends P {  
    int a = 100;  
    void show () {  
        sop(a);  
        sop (super. a);} →  
}
```



ABSTRACTION: -

Abstract Class: -

When we use abstract modifier along with any class then that class becomes abstract.

Now we cannot create its object so for assessing its property **it becomes mandatory to inherit such type of class by some non-abstract class.**

It may or may not contain abstract method.

Abstract Method: -

When we use abstract modifier along with any method then that method becomes abstract.

Now we cannot implement such type of method **it should be declare only.**

It must be in abstract class.

E.g., abstract void show;

Note: - It is compulsion for derived class of any abstract class to implement abstract method.

INTERFACE: -

Since java does not support multiple inheritance in terms of class which was the violation of OOPS concept so for overcoming from this violation interface is introduced and we can inherit multiple interfaces in class by using implements keyword.

Properties of interface: -

1. All variables inside the interface will be public, static and final by default.
2. It contains all abstract methods inside the interface will be public and abstract by default.
3. Interface may contain default and static method (JDK 1.8).
4. It may contain private as well as private static method.
5. We cannot create its object like abstract class.
6. It is compulsory for not abstract derived class of interface to implement all abstract methods of all interfaces.

Types of Interfaces: -

1. Normal Interface

2. Functional Interface: ➔

Only one abstract method.

E.g., Runnable, Cloneable, Comparable etc.

3. Marker Interface: ➔

No abstract method.

E.g., Serializable

TYPE CASTING: -

Two types: -

1. Implicit

E.g., `int x = 10;`

`Double d = x;`

2. Explicit

E.g.,

`Double d = 10.23;`

```
int x = (int) d;
```

1. Upcasting

E.g.,

```
Parent p = new Child ();
```

2. Down casting

E.g.,

```
Child c = (Child) new Parent ();
```

INNER CLASS (Nested Class): -

Outer class	Inner Class
Only default and public are use	All four-access specifier can be use
Can be Final, abstract but not static	We can use static with inner class

If we create any class inside any class is known as inner class or nested class.

We introduce this concept in java for strengthen the property of abstraction and encapsulation in java.

Properties: -

- Along with nested class we can use all types of access specifier.
- It can be static.

TYPES: -

1. Static inner class: -

2. Non static inner class: -

3. Anonymous inner class: - No class name. It can never be outer class.

LAMBDA EXPRESSION: - (Introduced in jdk 8)

Method without name.

It is an Anonymous function which is responsible to confine the code and it can be accessible by using functional interface.

E.g.,

Normal Method

```
public void show () {  
    Sop ("I'm normal method");  
}
```

Lambda Method

```
() -> {  
    Sop ("I'm now lambda");  
}
```

```
public int sum (int x, int y){  
    return (x+y);  
}
```

```
(x, y) -> x+ y;
```

```
public int square (int no){  
    for (int i = 1; i < 10; i++) {  
        Sop (no * i);  
    }  
    return no;  
}
```

```
(no) -> {for (int i = 1; i < 10; i++)  
{  
    Sop (no * i);  
    return no;  
}}
```

PACKAGE: -

It is the collection of certain class and interfaces which is responsible to do some specific type of task collectively.

It looks like a folder but has certain differences

- In the case of package, we need to declare folder name as a package name inside the file.
- It must be the first line of your source code.

- Declaration: - `package package_name;`

Uses: -

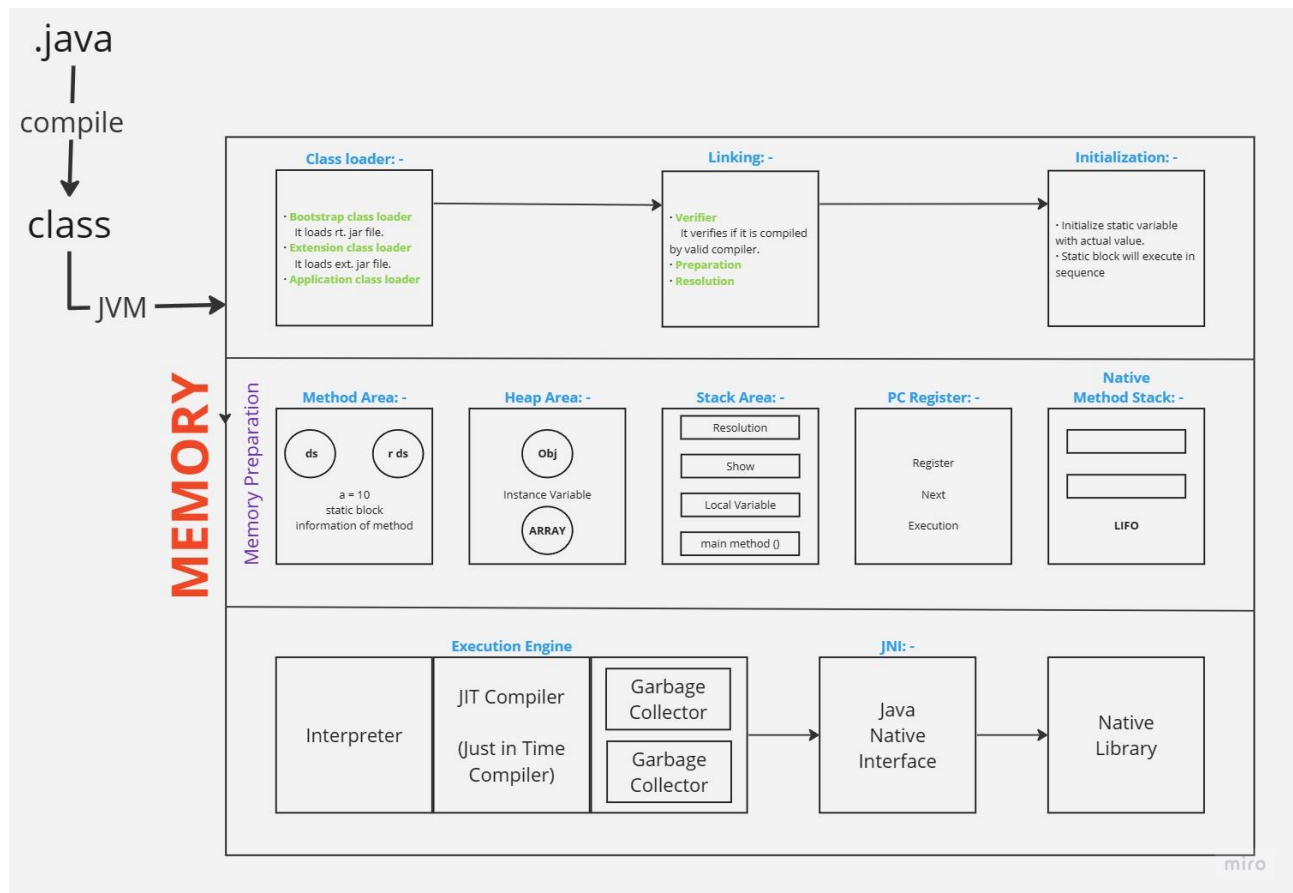
- It is useful to arrange file systematically at project level.
- It is useful to avoid many conflicts.
- It is useful to provide abstraction.

Protected can be access by anyone in package but outside the package only accessible by child.

Static import is responsible to provide all static properties of any class for direct use.
(Without using classname we can use its properties).

JVM ARCHITECTURE: -

Stages of JVM after receiving class file: -



Garbage collector: -

It is the mechanism provided by JVM which is responsible to free space in memory which is created by reference variable and it invoke automatically by JVM at particular time intervals.

Can be call excessively using `System.gc ();`

finalize (): - (अंतिम इच्छा)

It is use to free foreign resources. **It is callback method.**

E.g.,

```
finalize () {
```

```
    con. close (); // is a method in Java that is used to close a database connection.
```

```
    file. close ();
```

```
}
```

STRING: -

C++ → It is an array of character.

OOPS → It is sequence of character.

Java → It is a final class in java which define it as a sequence of character.

Properties: -

1. It is **immutable** by nature

Non-Changeable. It creates a memory scape which cannot be overwritten.

2. It is **synchronized** by nature.

Two Ways for defining string: -

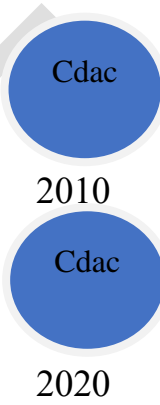
1. String Literal

String str = "Soft Polynomials";

2. By creating object using new keyword

String str = new String ("Soft Polynomials");

Note: - If we create an object of string using **new** keyword first time then two object are created physically in memory one in normal **Heap memory** and second in **String pool**.

Heap Memory		Stack	
	String Constant Pool		
	Cdac	s3	1010
		s2	1010
		s1	2020
		s	2010

```
String s = new String("Cdac");
String s1 = new String("Cdac");
String s2 = new String("Cdac");
String s2 = new String("Cdac");
String s3 = new String("Cdac");
s == s1 = false;
s == s2 = false;
s2 == s3 = True;
```

String constant Pool: -

It is the reserved memory area of heap memory which is created by string literals. In this memory area duplicity of object is not allowed so that we can use memory more efficiently.

STRING	STRING BUFFER	STRING BUILDER
Differences: -		
Immutable	Mutable	Mutable
Synchronized	Synchronized	Non-synchronized
Slow performance	Slow performance	Fast performance
Methods: -		
String s1 = "Ramesh"; String s2 = "Kapgate"; s1 + s2; String s3 = s1.concat(s2); Sop(s1); //Ramesh Sop(s2); //Kapgate	s1.append(s2); Sop(s1); // Ramesh Kapgate s1.reverse (); Sop(s1); //hsemaR	

Sop(s3); //Ramesh Kapgate	
------------------------------	--

indexOf () } Searching
lastIndexOf () }

indexOf () →

String str = "My name is Khan. I'm not terrorist. I'm Sharukh Khan one of the Bollywood actors."

← lastIndexOf ()

```
int x = str.indexOf ('K'); //11
      str.indexOf ('K',12); //25
      str.indexOf ("Khan"); //11
```

```
str.charAt (11); //K
```

String s [] = str.split ("");// to find number of words

char a [] = str.toCharArray (); // it will give character

s = " ";

str.isEmpty (); // false as it has blank

str.isBlank (); // true because there is nothing in it

} Gives boolean as answer

s1 = "Anup";

s2 = "anup";

Sop (s1. compareTo (s2)); // -32 ASCII value 65 – 97 = -32

s1. compareToIgnoreCase (s2); // 0 as it ignore the case

WRAPPER CLASSES: -

Primitive	Wrapper Class
int	Integer
byte	Byte
short	Short
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

In java all eight primitive data types have their corresponding build-in classes known as wrapper class.

It is useful to represent primitive data in object format.

To convert one data type to another we use parsing method.

E.g.,

Upto jdk 5;

```
int i = 10;
```

```
Integer x = new Integer(i);    //Primitive → Reference = Boxing
```

```
int a = x.intValue();         //Reference → Primitive = Unboxing
```

After jdk 5

```
Integer x = i;                // Auto Boxing
```

```
int a = x;                     // Auto Unboxing
```

Boxing: -

Conversion of primitive data to its corresponding to reference type is known as boxing.

Unboxing: -

Conversion of reference type data to its corresponding to primitive is known as boxing.

Constant Pool: -

valueOf () method is use.

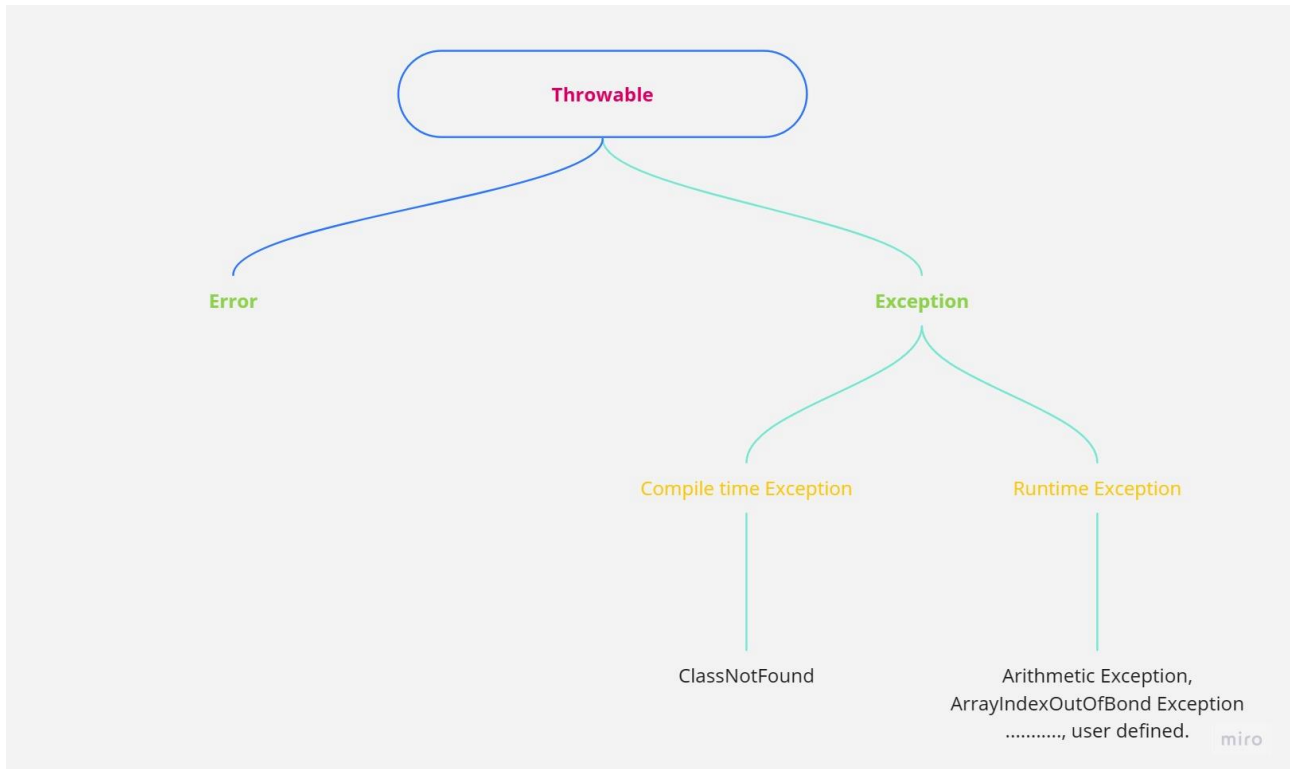
Family		Range
byte, short, int, long	→	-128 to 127
float, double	→	Constant pool is not available
boolean	→	true/false
char	→	'\u0000' to '\u007f'

EXCEPTION: -

Difference between error and exception: -

Exception	Error
<ul style="list-style-type: none">It is the problem which is generated due to faulty logical code.	<ul style="list-style-type: none">It is system generated problem.
<ul style="list-style-type: none">It can be handled by the programmer.	<ul style="list-style-type: none">It cannot be handled by the programmer. It must be corrected.
E.g., Arithmetic exception	E.g., Syntax error

Hierarchy of Exception: -



NOTE: - We can create our own exception by inheriting either throwable class or exception or runtime exception class

Type of Exception: -

1. Checked (Compile time) Exception: -

It is the exception which is recognized by our compiler and does not allow to compile the code is called compile/checked exception.

E.g., ClassNotFoundException, SQLException

2. Unchecked (Runtime) Exception: -

It is the exception which is recognized by our compiler and allow to compile the code but it generates problem at runtime is called Runtime/Unchecked exception.

E.g., Arithmetic Exception, ArrayIndexOutOfBoundsException

EXCEPTION HANDLING: -

In java we have 5 keywords to handle exception: -

Note: - It will be a part of method

1. try block: -

Inside try block we put the suspicious code in which exception may occur. In case there is an exception then this block will terminate and does not allow to execute the remaining code of try block.

Syntax: -

```
try {  
    -----  
    -----  
}
```

2. catch block: - (Same as if else statement)

This block will execute if any exception occurs in its corresponding try block. Generally, it is useful to provide substitute execution and for notifying to user that which type of exception is occur.

Syntax: -

```
try {...}  
catch (Exception e) {  
    -----  
    -----  
}
```

3. throws: - (Temporary handling)

It is the substitute of try and catch block for handling exception but it handles exception temporary whenever we call that code where exception is available exception regenerated at the time of calling.

Syntax: -

```
void show () throws Exception
{
    -----
}
```

4. throw: -

This keyword is responsible to generate exception manually. It is usually [useful to generate user-defined exception](#).

Syntax: -

```
void show () {
    throw new Exception ("msg");
}
```

5. finally: -

This block will execute where the exception is occurred or not in both the cases. It is useful to free the foreign resources.

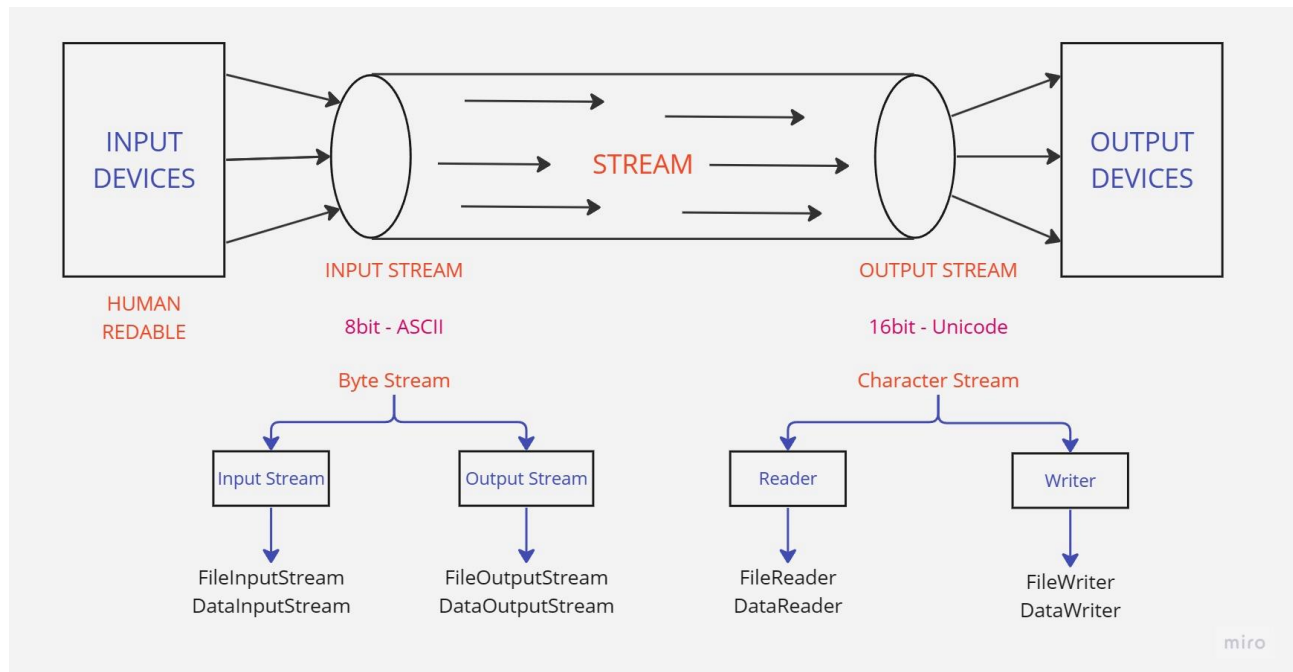
Syntax: -

```
finally {
    -----
}
```

INPUT OUTPUT (IO): -

ASCII: - 8 bit

Unicode: - 16 bit



Before Unicode flow of data was Stream.

After supporting Unicode Character Stream, it was also called as Byte Stream.

```
BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
```

```
br. read ();
```

```
br. readLine ();
```

SERIALIZATION: -

It is the process which is responsible to persist the state of object in secondary memory (file, database) for this we need to inherit serializable interface.

DATE, TIME, DATETIME, CALENDER: -

(to String, hashCode, equals)

They are legacy class and comes under java. util package. Where used before jdk8.

Now we use java. time package

localDate

localDate time

COLLECTION FRAMEWORK: -

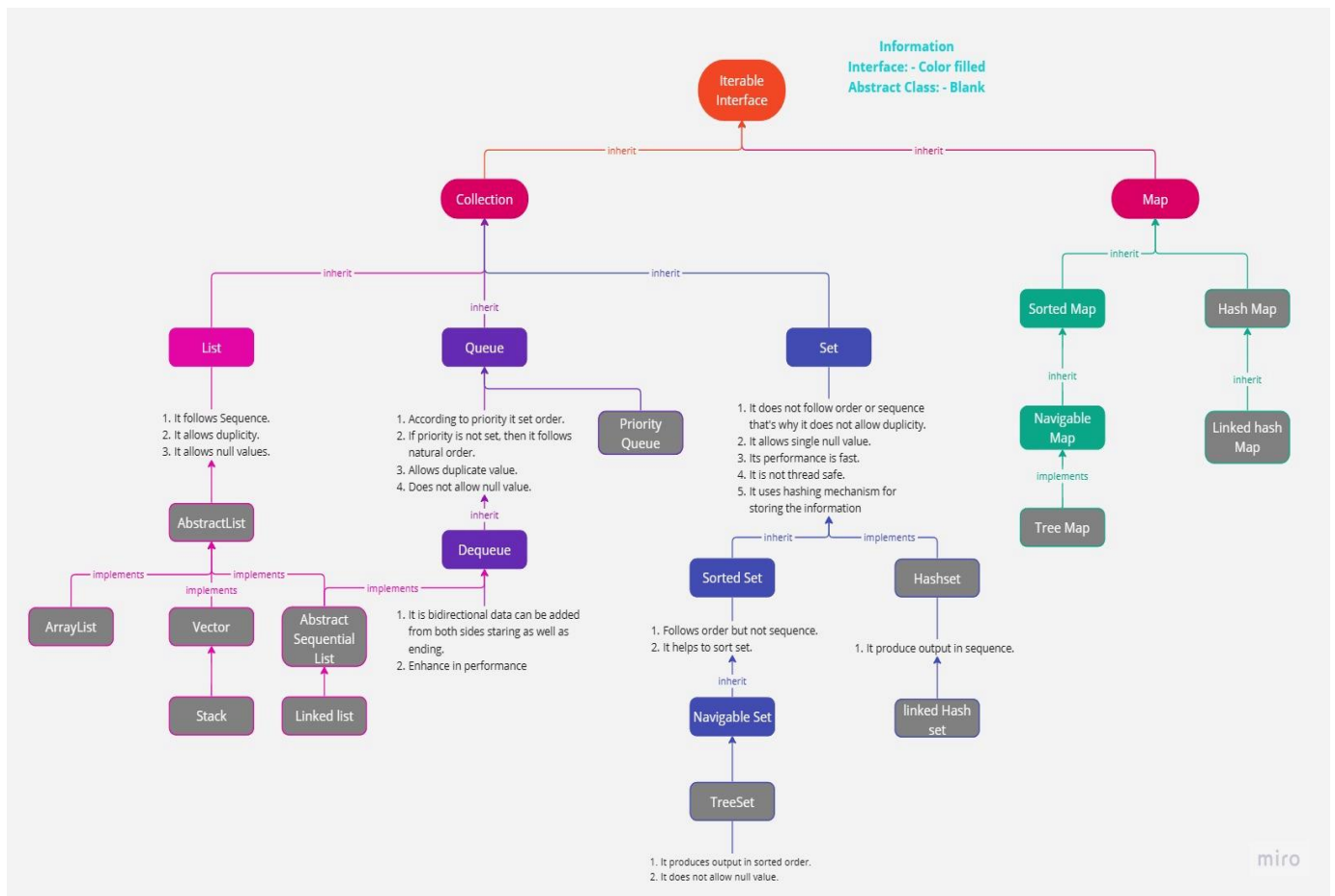
This framework is responsible is responsible to collect different type of data in form of object also known as elements dynamically.

It does not support primitive data. To store it we have to convert it into object using wrapper class.

Present in java. util package.

Three types: -

1. Interface
2. Abstract
3. Conacred



Array list	Vector	Stack	LinkedList
Common Properties: - <ul style="list-style-type: none"> • It is responsible to collect. • Follows the sequence. • It allows duplicity. • It allows null value. 			
Different Properties: -			
<ul style="list-style-type: none"> • It uses indexing for storing the data • It is non synchronized. 	<ul style="list-style-type: none"> • It is legacy class. (v1.0) • It is synchronized by nature. 	<ul style="list-style-type: none"> • It is synchronized. • It follows sequence of LIFO. 	<ul style="list-style-type: none"> • Instead of creating index it creates nodes. • It follows the mechanism of

<ul style="list-style-type: none"> • Its performance is fast for searching the data. • It performs slow for deletion and insertion of data 	<ul style="list-style-type: none"> • It uses indexing for storing the data • Its performance is fast for searching the data. • It performs slow for deletion and insertion of data 	<ul style="list-style-type: none"> • It supports indexing 	doubly linked list. <ul style="list-style-type: none"> • Addition and removal process becomes fast. • Performance for accessing the data is slow.
--	---	--	---

For sorting we have two interfaces: -

1. Comparable: -

For natural Sorting and available for only one column.

It has method called **compareTo (o)**.

2. Comparator: -

Can be use on “n” number of columns.

It has method called **compare (o1, o2)**.

Concurrency Collection: -

Since most of the data structure in collection is supported multithreading (it is non synchronized) that's why if multiple threads try to manipulate any data structure it may create unsafe operation and data may corrupted so for making it safe, we have the facility of concurrent collection.

THREAD: -

It is the path of execution of code.

Difference between thread and process: -

Process

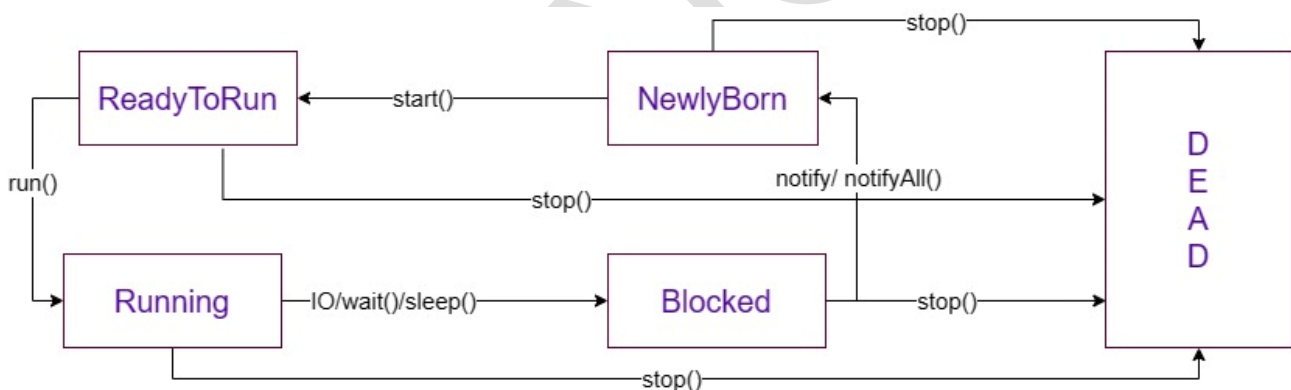
- Collection of thread
- It is heavy weighted
- Every process occupies its own memory space in the system

Thread

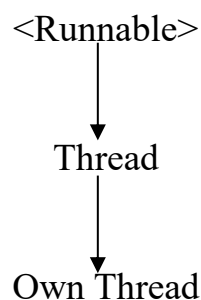
- It is the path of execution of code.
- It is light weighted
- Thread does not occupy its separate memory space. It simply shares the same space which is occupied by its corresponding process.

Life cycle of thread: -

When thread is born it is called NewlyBorn.



Hierarchy of thread: -



Method of threads: -

start (): -

It is the method of thread class which is responsible to set newly born thread to readToRun stage.

run (): -

It is only method of runnable interface. In this method we need to put the code to which we need to assign thread. It will run automatically when start method is called.

sleep (): -

It is the method of thread class in which we need to pass time in milliseconds and it is responsible to block the thread for given time. The thread blocked by this method invoke automatically after completion of time.

wait (): -

It is the method of object class which is responsible to block the thread for indefinite time. So, for invoking such type of blocked thread we need to call either notify/notifyAll method.

notify / notifyAll method: -

These are the methods of object class which are responsible to invoke blocked thread.

wait (): -

It is method of thread class and responsible to achieve dead stage of thread.

Other method of threads: -

isAlive (): -

This method is responsible to check whether the thread is alive or not.

suspend(): -

It is the method of thread class which is responsible to suspend the running thread.

resume(): -

It is the method of thread class which is responsible to resume suspended thread.

yield (): -

It is the method of thread class when we called this method on any running thread then that thread paused itself momentarily and allow other waiting thread for execution. After completion of execution of allowed thread, it invokes itself automatically.

join (): -

It is recommended that main thread should not exit before child thread so for achieving this recommendation we called join method on child thread than it become compulsion for main thread that it cannot exit itself without exiting child thread on which join method is called.

We have two types of thread: -

1. **Demon thread:** - All system defined thread belongs to this category.

E.g., main thread, thread of garbage collector.

2. **Non-Demon thread:** - All user define thread belongs to this category.

PRIORITY: -

IT is on scale 1-10 always.

1 = minimum.

5 = normal (by default).

10 = maximum.

E.g.,

a → 7

b → 5

c → 3

a. `setPriority (7);`

```
a. setPriority (Thread. MAX_PRIORITY -3);    // 7
c. setPriority (b. getPriority () - 2);      // 3
```

SINGLE THREADING / SYNCHRONIZATION / THREAD SAFETY: -

It is the process in which any thread can execute safely in multi-threaded environment. It is achieved by using **synchronized** keyword.

E.g.,

```
synchronized void Show () {.....}
```

Or

```
void Show () {
    .....
    synchronized (this) {
        .....
    }
    .....
}
```

DEADLOCK: -

If multiple threads have lock on its corresponding resources and waiting for releasing other thread's lock on its resource so that they can use it but if there is condition if all threads waiting for each other simultaneously to release their lock in that situation deadlock condition is arrived.

GENERIC: -

REFLECTION API: -

Use to do ethical hacking.