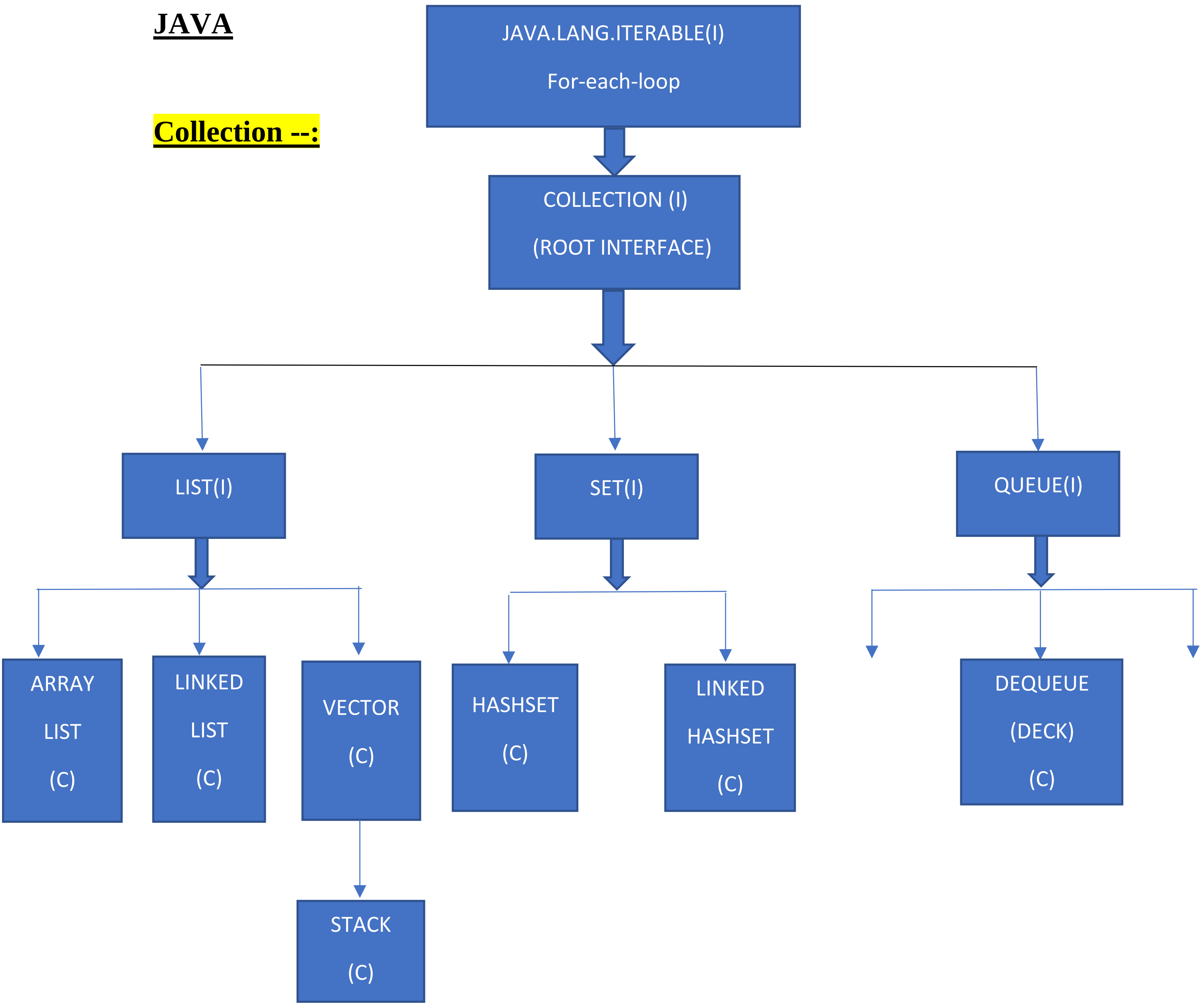
JAVA

Collection --:



LIST--:

1. The list is an indexed Sequence
2. List ALLOW duplicate element (non-unique)
3. Element by their position can be accessed
4. Multiple NULL elements can be stored
5. List implementations are {Array List, LinkedList, Vector →Stack}

6. Methods are used in

   Iterator = iterator (); ➔ Forward direction

   ListIterator = listIterator (); ➔ Forward direction as well as Backward direction

```
ArrayList <Integer> list=new ArrayList <Integer> ();

ListIterator <Integer> itr =list. listIterator ();

while(itr.hasNext())

{

                int element=itr.next();

                System.out.println(element);

}

while(itr.hasPrevious())

{

                int element=itr.previous();

                System.out.println(element);

}
```

SET--:

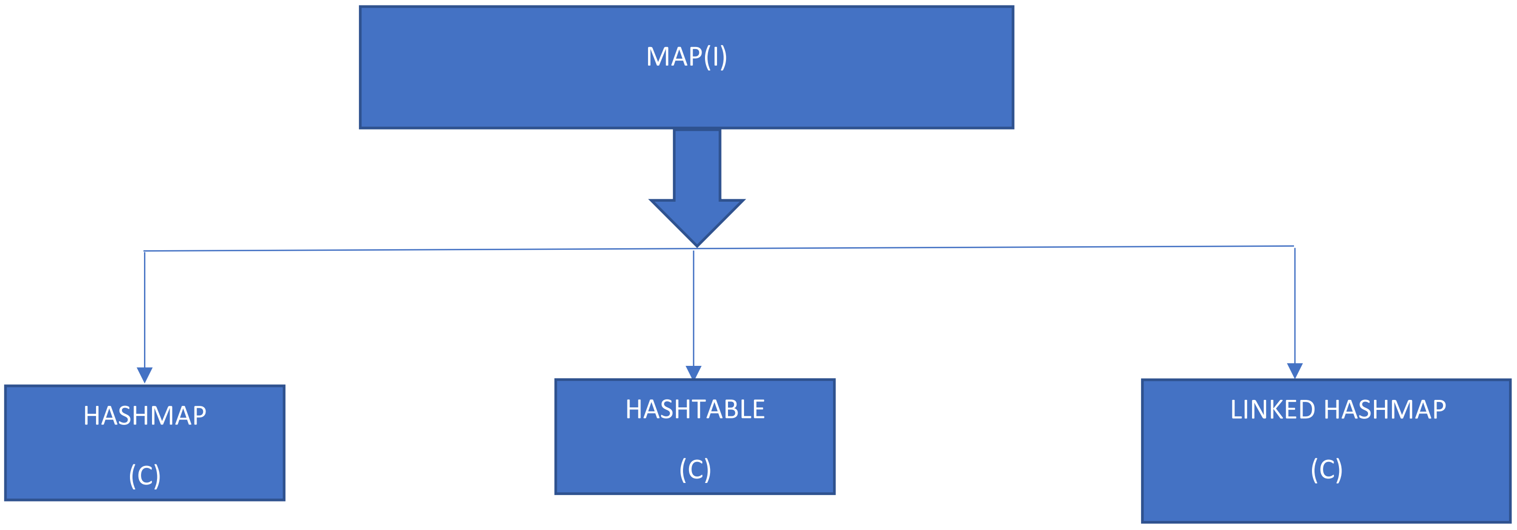
1. The set is a non-indexed sequence
2. Set does NOT allow duplicate elements (Unique)
3. Position access of element is NOT allowed
4. ONLY ONE NULL element can be stored
5. SET Implementation {HashSet, LinkedHashSet, TreeSet}
6. TREESET is Sorted set.
7. Methods are used in
   Iterator = iterator ();

HOW MANY COLLECTIONS BY DEFAULT SYNCHRONIZED??

Thread Safe classes are given below --:

- ⇨ 1. VECTOR
- ⇨ STACK ➔ SUB CLASS OF VECTOR
- ⇨ 2. HASH TABLE
- ⇨ PROPERTIES ➔ SUB CLASS OF HASH TABLE

MAP <K, V>



MAP (Hashing Technique)

Map is pair of KEY-VALUE

1. You can store duplicate values but You cannot store duplicate key
2. Insertion order is NOT preserved. Can have NULL key(ONLY ONE) and values(MULTIPLE).

Non- final class must be override/implement

equal (); and hashcode (); Methods

➔hashcode is used for fast searching.

TREEMAP --:(RED BLACK TREE) FIRST ENTRY WILL BE NULL Otherwise NULLPOINTEREXCEPTION

KEY CANNOT NULL

VALUE CAN NULL

HASH TABLE → does not allow null values

EXCEPTIONS-:

How to Handle →

1. Try → inspect set of parameters
2. Catch → handle exception
3. Throw → Raised and Generate Exception
4. Throws →Forward Exception
5. Finally →Execute at last

Exceptions	Error
➔ It is a subclass of java.lang.Throwable	➔ It is a subclass of java.lang.Throwable
➔ If runtime error gets generated due to application, then it is called Exception EX -: ArrayIndexOutOfBound, Null Pointer Exception	➔ If runtime error gets generated due to environmental condition, then it is called Error EX -: Internal Error, Virtual Machine Error, StackOverFlowError
➔We can write try catch to handle exception	➔We can write try catch to handle error
➔We can recover from Exception	➔We cannot recover from Error
➔It is recommended to handle	➔It is NOT recommended to handle

CHECKED EXCEPTION	UN-CHECKED EXCEPTION
Java.lang.Exception and all its sub classes are considered as checked exception class.	Java.lang.RunTimeException and all its sub classes are considered as unchecked exception class.
It is mandatory to handle checked exception	It is NOT mandatory to handle unchecked exception

Example: ClassNotFoundException IOException	Example: NumberFormatException ClassCastException NullPointerException ArrayIndexOutOfBounds

NOTE→

1. Rules of overriding is NOT applicable for unchecked exception.
2. Rules of overriding is applicable for checked exception.
3. Generate the Exception through another Exception is called → Exception Chaining.

➔Check exception is handle by java compiler that's why is called compile time error.

➔Unchecked exception is handle at run time.

OBJECT CL ASS --:

Object class declare in java.lang.package.

Java.lang.object ultimate super/root of java class hierarchy.

It contains only parameter-less constructor. (Default constructor).

It contains 11 methods (5 non-final and 6 final method) ➔

NON-FINAL

1. Public string tostring ();
2. Public Boolean equal (object obj);
3. Public int hashcode (); (NATIVE)
4. Protected object clone (); throw cloneNotSupportedException (NATIVE)
5. Protected void finalize (); throw throwable

FINAL

1. Public final class <?> getClass (); (NATIVE)
2. Public final void wait (); throw InturruptedException
3. Public final void wait (long timeout); throw InturruptedException (NATIVE)
4. Public final void wait (long timeout, int nanos); throw InturruptedException
5. Public final void notify (); (NATIVE)
6. Public final void notify All (); (NATIVE)

Native → code/logic written in C or C++ language and use in java language.

INTERFACE →

Interface in the java is a blueprint of the class. It specifies what a class must do and not how.

1. It is used to achieved abstraction.
2. It supports multiple Inheritance.
3. It can be achieved to loose coupling.

Syntax

```
 interface InterfaceName
{
    Methods // abstract, public
    Void print (); //By default abstract
   Fields // public, static, final
   Default void display () {/* code */}
 static void display ()
     {
         System.out.println("hello");
     }
}
```

Abstract class	**Interface**
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.

4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

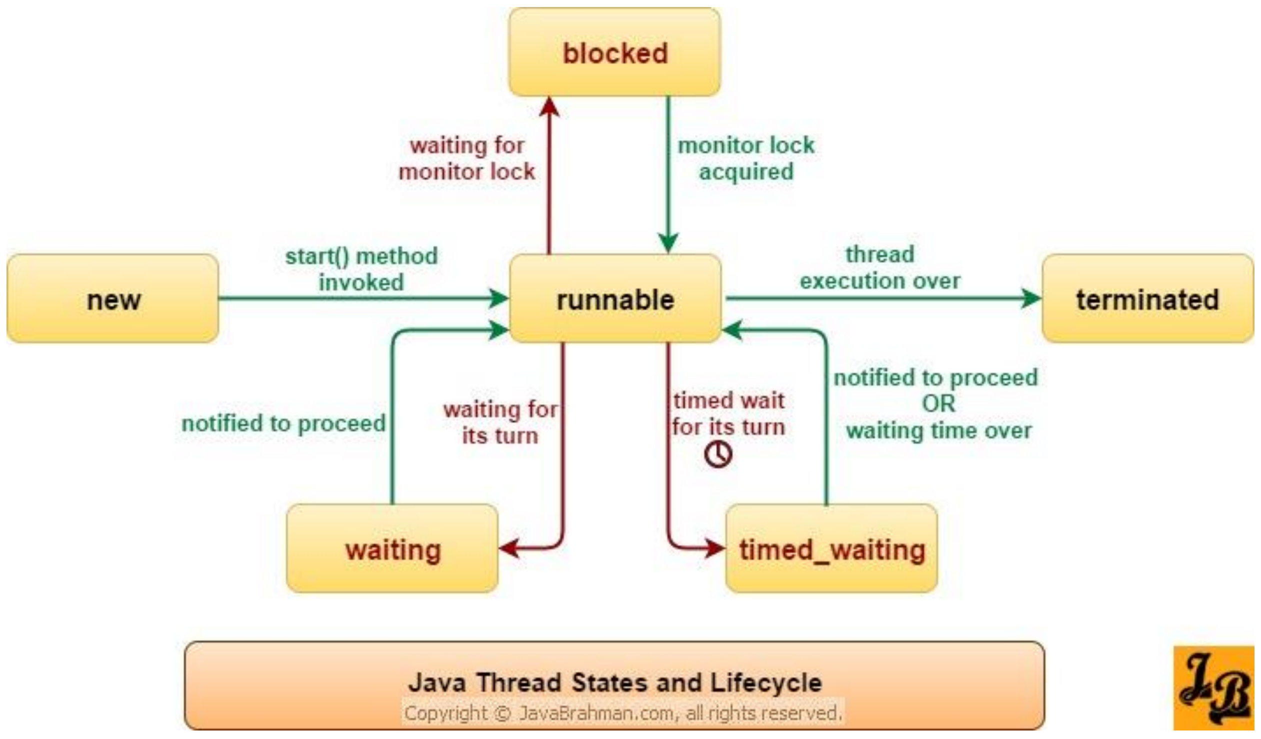
9)**Example:**
public abstract class Shape {
public abstract void draw ();
}
	Example:
	public interface Drawable {
	void draw ();
	}

Similarities between Abstract and Interface.

1. Does not support instantiation (Cannot create object)
2. Reference type
3. Both can define an abstract method

FUNCTIONAL INTERFACE → Only one method declare inside interface is called functional interface.

THREADING --: (STATE OF THREAD)



THREAD PRIORITIES --:

- ➪ Thread.MIN_PRIORITY =1
- ➪ Thread.NORM_PRIORITY =5 (default value)
- ➪ Thread.MAX_PRIORITY =10

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

`Void`	`run()` When an object implementing interface `Runnable` is used to create a thread, starting the thread causes the object's `run` method to be called in that separately executing thread.

Class Myclass implements Runnable {public void run () {}}

To invoke run method use ➔ Thread t=new Thread (); t.start ();

Q) Execution of a java thread begins on which method call?

➔ Start ()

On thread start method create a thread and execute it. Note that thread start () method calls run () method internally.

Q) How many ways a thread can be created in Java multithreading?

➔ 2

Threads in Java multithreading can be created in two ways. First, by implementing runnable interface and second as by extending thread class.

Q) Which statements is/are correct

➔On calling Thread start () method a new thread get created.

➔Thread start () method call run () method internally

Q) Which method is used to make main thread to wait for all child threads

➔Join ()

Q) Which method is used to wait for child threads to finish in Java?

➔Join ()

Q) If a priority of a java thread is 3 then the default priority of its child thread will be

➔3

The default thread priority of a child thread is same as what parent

Q) Which method is used to check if a thread is running?

➔isAlive ()

Q) True statement about process and thread is/are

1. **If a child process crashes all main process will also be crashed (wrong)**
2. **If a child thread is crash entire process will crash (true)**
3. **Threads have their own memory stack (true)**
4. **Each process has different virtual space (true)**

Q) Daemon thread runs in

1. **Background**

➔A *daemon thread* in Java is a thread that runs in the background within same process. Daemon threads are like Service providers for other threads running in the same process. Daemon threads are used for background support task like handling request or events etc. and are only needed while normal threads are running. Daemon threads work with normal/user threads.

Make a point that daemon threads are terminated by the JVM when there are no longer any user threads running, including the main thread.

Notes:

We need to understand two important Java APIs setDaemon () and isDaemon () while working with Daemon threads.

setDaemon (boolean value)- Set it to "true" to make thread as a daemon thread.

isDaemon()- to test if particular thread is Daemon.

Q) To create a daemon thread

➔First thread setDaemon() is called then start()

Q) Which will contain the body of the thread in Java?

➔Run ()

Q) Thread synchronization in a process will be required when

1. **All threads sharing the same address space**
2. **All threads sharing the same global variables**
3. **All threads sharing the same files**
4. **All** **Answer: 4**

Q) Which thread will be executed first if two threads have same priority

➔It depends upon operating system

Q) The life cycle of a thread in java is controlled by ➔JVM

Life Cycle of a thread

1. **New Thread:** When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state. When a thread lies in the new state, its code is yet to be run and hasn't started to execute.
2. **Runnable State:** A thread that is ready to run is moved to a runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run.
   A multi-threaded program allocates a fixed amount of time to each individual thread. Each and every thread runs for a short while and then pauses and relinquishes the CPU to another thread so that other threads can get a chance to run. When this happens, all such threads that are ready to run, waiting for the CPU and the currently running thread lie in a runnable state.
3. **Blocked/Waiting state:** When a thread is temporarily inactive, then it's in one of the following states:
   - Blocked
   - Waiting

4. **Timed Waiting:** A thread lies in a timed waiting state when it calls a method with a time-out parameter. A thread lies in this state until the timeout is completed or until a notification is received. For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.
5. **Terminated State:** A thread terminates because of either of the following reasons:
    - Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
    - Because there occurred some unusual erroneous event, like segmentation fault or an unhandled exception.

Marker Interface Example ➔

1. **Serializable**
2. **Cloneable**
3. **Remote**

A marker interface is an interface that **has no methods or constants inside it**. It provides **run-time type information about objects**, so the compiler and JVM have **additional information about the object**.

An interface which does not contain any member is called marker/tagging interface.

Marker interface are used to generate meta-data for the JVM.

Prior to JDK 1.5 marker interface were used to generate metadata.

Widening (Lower to Higher) (Implicitly) ➔

Process of converting value of variable of narrower type into wider type is called widening.

Ex. int a=4;

long b=a;

System.out.println(b);

Narrowing (Higher to Lower) (Explicitly) ➔

Process of converting value of variable of wider type into narrower type is called widening.

Ex. long a=4;

int b=(int)a;

System.out.println(b);

Auto-Boxing (Lower to Higher) ➔

Process of converting value of variable of primitive type into non-primitive type

Is called Auto-Boxing.

Un-Boxing (Higher to Lower) (Explicitly) ➔

Process of converting value of variable of non-primitive type into primitive type

Is called Un-Boxing.

Lambda Expression

* Expression is a statement which may contain:

    1. Literals / constants

    2. Variables

    3. Operators

* Example:

```java

    int num1 = 10, num2 = 20;
```

    int result = num1 + num2; //Arithmetic Expression

* "->" operator is called lambda operator. Its meaning is "goes to".

* If expression contains lambda operator then such expression is called lambda expression.

* Syntax:

    - FunctionalInterface ref = (Input params)-> { Lambda Body };

    - Input parameters goes to lambda body.

    - If lambda body contains single statement then use of { } is optional.

* Lambda expresion is also called as Anonymous method.

* Abstract method of Functinoal interface is called Functinoal method / method descriptor.

* To implement functional interface we should define lambda expression

* If interface contains single abstrasct method then it is called functional interface.

```java

@FunctionalInterface

interface Printable {

	void print (); //Functional method / Method descriptor

}

```

```java

Printable p = ( )-> System.out.println("Inside Lambda Expresion");
```

```
p.print();
```

* Consider another example:

```java
@FunctionalInterface
interface Printable {
    void print (String message);      //Functional method / Method descriptor
}
public class Program {    //Program.class
    public static void main(String[] args) {
        //Printable p = ( String msg )-> System.out.print(msg);
        //Printable p = ( String message )-> System.out.print(message);
        //Printable p = ( message )-> System.out.print(message);
        Printable p = message -> System.out.print(message);
        p.print( "Have a nice day." );
    }
}
```

* Consider another example

```java
@FunctionalInterface
```

```java
interface IMath{
    int sum( int num1, int num2 );
}
public class Program {    //Program.class
    public static void main(String[] args) {
        IMath m = ( num1, num2 )-> num1 + num2;
        int result = m.sum(10, 20);
        System.out.println("Result    :    "+result);
    }
}
```

* Consider another example:

```java
@FunctionalInterface
interface IMath{
    int factorial( int number );
}
public class Program {    //Program.class
    public static void main(String[] args) {
        IMath m = number ->{
            int fact = 1;
```

```
                    for( int count = 1; count <= number; ++ count ) {
                            fact = fact * count;
                    }
                    return fact;
            };
            int result = m.factorial(5);
            System.out.println("Result        :        "+result);
    }
}
```

<u>### String Handling</u>

* In Java, string is a collection of character object.

* If we want to manipulate String then we can use:

   1. java.lang.String

   2. java.lang.StringBuffer

   3. java.lang.StringBuilder

   4. java.util.StringTokenizer

String

* It is a final class declared in java.lang package.

* It implements Serializable, Comparable<String>, CharSequence interface.

* String is not buit-in / primitive type in Java. It is a class hence it is considered as reference type.

* We can create String instance with and without new operator.

* Consider example:

```java

String s1 = new String("SunBeam");  //OK : String Instance => Heap

```

```java

String s2 = "SunBeam";  //OK : String Literal => String Literal Pool

```

* String s2 = "SunBeam" is equivalent to

```java

char[] data = {'S','u','n','B','e','a','m'};

String str = new String( data );

```

* String objects are constant. If we try to modify its state then new string instance gets created. In other words, String objects are immutable objects.

* If we want to concatenate strings then we should **use "concat()"** method.

* If we want to concatenate string and variable of primitive/non primitive type then we should use operator.

StringBuffer and **StringBuilder**

* Both are final classes declared in java.lang package.	* Both are final classes declared in java.lang package.
* Both are used to create mutable String object.	* Both are used to create mutable String object.
* It is mandatory to use new operator to create their instance.	* It is mandatory to use new operator to create their instance.
* Even though both are final, equals and hashCode() method is not overriden inside it.	* Even though both are final, equals and hashCode() method is not overriden inside it.
* StringBuffer is thread-safe/synchronized	StringBuilder is Not thread-safe/ unsynchronized.
StringBuffer is introduced on JDK 1.0	StringBuilder is introduced in JDK 1.5

Local Class

* We can define class inside scope of another method. It is called local class/method local class.

* We cannot use reference/instance of method local class outside method.

* Types of Local class:

1. Method Local Inner class

2. Method Local Anonymous Inner class

Method Local Inner class

* In Java, we cannot declare local variable/class static.

* Non static, method local class is also called as method local inner class.

Method Local Anonymous Inner class

* In Java, we can define class without name. It is called anonymous class.

* In Java, we can create anonymous class inside method only hence it is called as method local anonymous class.

* In Java, we cannot declare local class static. Hence anonymous class is also called as method local anonymous inner class.

```java
class Program{  //Program.class
    public static void main(String[] args) {
        //Method Local Class
        class Complex{  //Program$1Complex.class
    }
 }       }
```

NON-STATIC AND STATIC INNER CLASS --: (**Outer$Inner.class**)

NON-STATIC	STATIC
Non-Static Nested class is also called as inner class.	When we declared nested class static then it is simply called as static nested class.
If implementation of nested class depends on top level class then nested class should be non-static	If implementation of nested class do not depends on top level class then nested class should be static.
Instantiation of top-level class: Outer out = new Outer ( );	Instantiation of top-level class: Outer out = new Outer ( );
Instantiation of non-static nested class:	Instantiation of static nested class:

First way --:
 	
Outer out = new Outer ();
 	
Outer.Inner in = out.new Inner ();
 	
Second way --:
 	
Outer.Inner in = new Outer ().new Inner( );
	Outer.Inner in = new Outer.Inner( );

STATIC INITIALIZATION BLOCK AND CONSTRUCTOR --:

STATIC INITIALIZATION BLOCK (SIB)	**CONSTRUCTOR**
SIB is design to initialize static field	Constructor design to use non-static field
Gets call once per class	Gets call once per object (Instance)
We cannot call one static block to another static block.	We can call constructor from another constructor.

" SUPER KEYWORD " ➔

" Super " keyword is a reference variable which is used to refer

Immediate parent class object.

We can call super class methods and constructor from subclass methods.

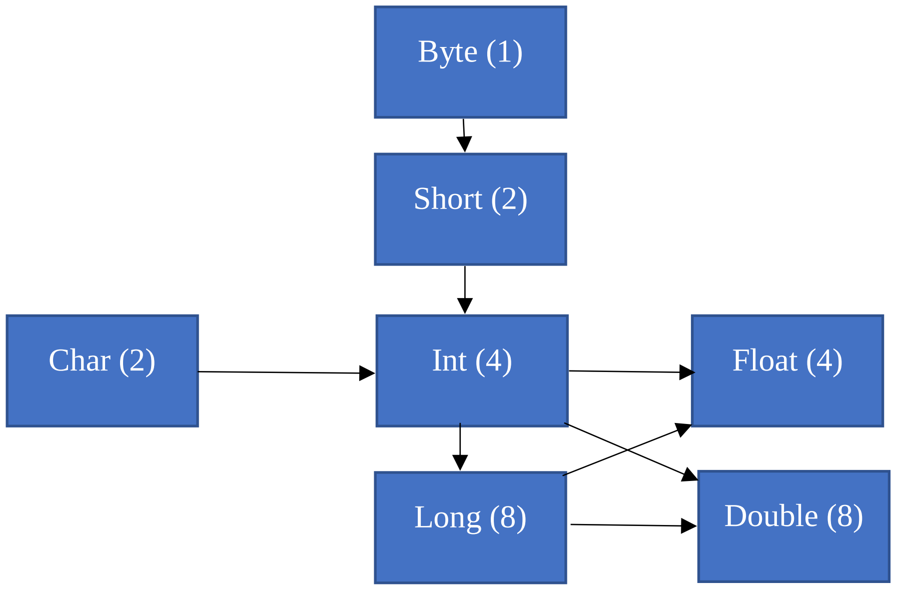
This keyword ➔

If we call method on instance then compiler implicitly pass reference of current instance as an argument and to store the argument compiler implicitly declare parameter in method is called this reference.

If local variable name and field variable name is same then use before field variable.

AUTOMATIC PROMOTION --:

One type is promoted to another implicitly if no matching data type is found.



ENUM --:

An Enum can, just like a class , have attributes and methods. The only difference is that Enum constants are public , static and final (unchangeable - cannot be overridden). An Enum cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces).

If we want to improve readability of the source code then we should use enum.

In Java, using enum, we give name to any literal or group of literals.

We cannot change ordinal of enum constant. enum is a keyword in Java.

enum is non primitive/reference type in Java.

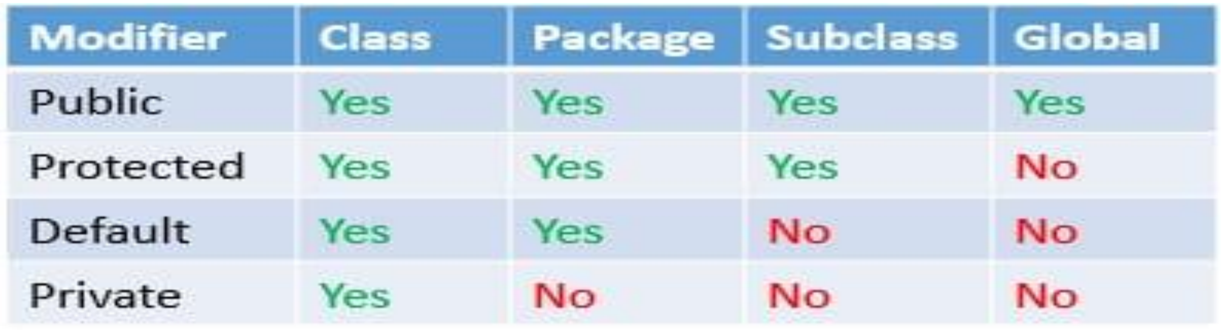
Compiler generate .class file per interface, class and enum. Compiler generated code for enum.

If we define enum in Java, then it is implicitly considered as final class. Hence we cannot extends enum. Enum constants are references of same enum which is considered as public static final field of the class.

values() and valueOf() are methods of enum which gets added at compile time.

In Java, if we want to assign name to the literals then it is mandatory to define constructor inside enum. And to get value of the literal it is necessary to define getter methods inside enum.

ACCESS MODIFIER --:



Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

DIFFERENCE BETWEEN FINAL, FINALLY, FINALIZER --:

FINAL →

- It is a keyword.

1) Once declared, final variable becomes constant and cannot be modified.
(2)final method cannot be overridden by sub class.
(3) final class cannot be inherited.

FINALLY →

- It is a block.

(1) finally block runs the important code even if exception occurs or not.
(2) finally block cleans up all the resources used in try block

If we don't want to execute Finally block ?? then??

terminate the JVM before control is going to finally the JVM does not execute Finally block. **System.exit (0);**

FINALIZER →

- It is a method.
- finalize method performs the cleaning activities with respect to the object before its destruction.

SERILIZATION AND DESERILIZATION --:

Interface for control over serialization and deserialization =➔ EXTERNALIZABLE

SERILIZATION	**DESERILIZATION**
Serialization is a mechanism of converting the state of an object into a byte stream	Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.
For serializing the object, we call the writeObject() method of ObjectOutputStream class	for deserialization we call the readObject() method of ObjectInputStream class.
It mainly used in hibernate Jms technique	This mechanism is used to persist the object
Java Instance -----------> Binary data	Binary data -------> Java Instance

If we don't want to serialize any FIELD then we should declare.

- ⇨ Static
- ⇨ Transient

When to use serialization??

➔We can use transfer data between different distributed system.

JDB ➔ debugger

InstanceOf ➔ The instanceof operator in Java is **used to check whether an object is an instance of a particular class or not**.

Java Source code ➔ .java

Java byte code ➔ .class

length ➔use with Array

length () ➔ use with String

FILE HANDLING ➔

Que 1: Which of this class contains the methods used to write in a file?

- ➔ FileInputStream

Que 2: Which of this exception is thrown in cases when the file specified for writing is not found?

- ➔ FileNotFoundException

Que 3: Which of these methods are used to read in from file?

➔ Read()

Que 4: Which of these values is returned by read() method is end of file (EOF) is encountered.

➔ -1

Que 5: Which of this exception is thrown by close() and read() methods?

➔ IOException

Que 6: Which of these methods is used to write() into a file?

➔ write()

Que 7: Which of the following packages contain classes and interfaces used for input & output operations of file?

➔ java.io

Que 8: Which of this class is not a member class of java.io package?

➔ String

Que 9: Which of the following is method for testing whether the specified element is a file or a directory?

➔ isFile()

Que 10: A stream is a sequence of data and stream is composed of

➔ Bytes

Que 11: SequenceInputStream class is used to read data from

➔ c) Multiple streams

Que 12: Which class can be used to read data line by line using readLine() method?

➔ BufferedReader

Que 13: Which are the different ways to read data from the keyboard?

a) Scanner b) InputStreamReader c) DataInputStream d) All of given

Que 14: Which of the following is used an internal buffer, It adds more efficiency than to write data directly into a stream. So, it makes the performance fast?

➔BufferedOutputStream

Que 15: Which are subclasses of FilterInputStream and FilterOutputStream?

➔Both DataInputStream and DataOutputStream

Que 16: When does Exceptions in Java arises in code sequence?

➔Run Time

Que 17: Which of these keywords is used to manually throw an exception?

➔ throw

Que 19: Which exception is thrown when divide by zero statement executes?

➔ ArithmeticException

Que 21: Which of these exceptions will be thrown if we declare an array with negative size?

➔NegativeArraySizeExeption

Que 22: Which of these exceptions will be thrown if we use null reference for an arithmetic operation?

➔NullPointerException

Que 23: Which of these clauses will be executed even if no exceptions are found?

➔finally

Que 24: Which of these exceptions will occur if we try to access the index of an array beyond its length?

➔ArrayIndexOutOfBoundsException

Que 25- Which of these classes are the direct subclasses of the Throwable class?

➔ c) Error and Exception class

Sr. No.	Key	IOC	Dependency Injection
1	Design Principle	It is design principle where the control flow of the program is inverted	It is one of the subtypes of the IOC principle
2	Implementation	IOC which is implemented by multiple design patterns service locator , events , delegates and dependency Injection	DI is design pattern which can be achieved by constructor and setter injection
3	Use Case	Aspect oriented programing is one way to implement IOC	In Case of change in business requirement no code change required