Core Java

High-Level Language	Low-level language
It can be considered as a programmer-friendly language.	It is considered as a machine-friendly language.
It requires a compiler/interpreter to be translated into machine code.	It requires an assembler that would translate instructions.
It can be ported from one location to another.	It is not portable.
It is easy to understand.	It is difficult to understand.
It is easy to debug.	It is difficult to debug.
It is less memory efficient, i.e., it consumes more memory in comparison to low-level languages.	It consumes less memory.

JRE - Java Runtime Environment

JVM - Java virtual Machine

JDK - Java Development kit

JVM

API - Application Programming interface

Source code. Compile. Byte packing .jar. .war

500

Shoppingcart.jar

Shoppingcart.war- web server, - Apache tomcat or Jetty

SIT - developer

UAT - QA

PROD - user

Development -> Bitbucket or SVN or GIT - Jules/Jenkins- AIM - Deploy

Master
Develop -SIT
Feature. PR
BugFix
Release 2022.02.02 - UAT and PROD

Sequential Execution
T1 - 10
T2 - 10
T3 - 10

30

Concurrent Execution
T1 - 10
T2 - 10
T3 - 10

10

Daemon Thread - Background

User Thread - Foreground

Garbage Collector

- For memory deallocation

- System.gc()

Employee extends Object
{

Public void finalize() {

}

Throughput time

Object

www.amazon.in

Structure of Java Project
_____

JavaProject

- package - in.amazon.admin
    - Java Class - EmployeeMain.java
- package - in.amazon.tx
- package - in.amazon.user

Admin
Tx
User

Project - First char upper case and second String first char upper case
- - in.amazon.admin - lower case
    - class
    - class

- package. - in.amazon.tx

- package. - in.amazon.user

Whats package?
- group of related classes and interfaces
- code maintainability

Access Modifier:
1. Public
2. Private
3. Protected
4. Default

Public class ClassName {

// variables /field

//constructors

// methods/functions

}

1. Create Bank and Contact
2. BankMain and ContactMain
3.

Constructor
———————
- Its use to initialise the object of the class
- It should be public

- We should not use final or abstract
- Types of constructor
  - Default
  - Parameterized

Method
———————

1. Static method
2. Instance method

Primitive Data Type
———————————————

Private long empId=10l;
Private float I= 10.0f;

- To allocate the memory space

1. byte - 8 bit
2. short - 16 bit
3. int. - 32 bit
4. long - 64 bit
5. float - 32 bit
6. double - 64 bit 1432413432.00
7. boolean - 1 bit. True/false
8. char - 16 bit

Wrapper Class

Byte
Short
Integer
Long
Float
Double
Boolean
Character

Autoboxing and Unboxing
———————————————

- Convert primitive data type to the corresponding wrapper class - auto boxing. Int — Integer
- Convert wrapper class to corresponding primitive data type - unboxing Integer - int
-

String

_____

- String is not primitive data type
- String is a class
- String is Immutable class

String str = “hi”;

String str1 = str.toUppercase();

sout(str) ; //hi
sout(str1); // HI

Create a new String
_____

1. String literal
2. Using new Operator

1. String literal

Private String str = “welcome”;

String pool

String str1 = “welcome”;

2. Using new Operator - heap

String str = new String(“welcome”);

String str1 = new String(“welcome”);

Str1.intern();

String Comparison
_____

1. ==
2. equals()
3. compareTo()

String input1 = “welcome”;

String input2 = new String (“welcome”);

1. == compare the reference of both the string or memory address of both the string

```
If (input1 == input2) {
        return true;
} else {
        return false;
}
```

O/p : false

2. equals() - Boolean , compare the content of both the string.

equalsIgnoreCase()

```
If (input1.equals(input2)) {
        return true;
} else {
        return false;
}
```

O/p : true

3. CompareTo () - 0, +ve, -Ve

- Its similar to equlas()method
- Compare the content of the String

```
If (input1.compareTo(input2) == 0) {
        return true;
} else {
        return false;
}
```

O/P : true

Mutable String

- We can modify the string
- If you are making any changes it will update on the same string

1. StringBuilder - Not Thread safe
2. StringBuffer - Thread safe

T1 T2 T3

Operators

_____

Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.

A + B

1. Assignment Operator

    - Assignment operator "="
    - It assigns the value on its right to the operand on its left:

```
 int cadence = 0;
 int speed = 0;
 int gear = 1;
Employee emp = new Employee();
```

    - This operator can also be used on objects to assign object references,

2. Arithmetic Operator

    - It provides operators that perform addition, subtraction, multiplication, division and modulo division

Operator	Description
+	Additive operator (also used for String concatenation)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Remainder `operator

```
Int a = 10;
Int b = 5;

Int c= a % b;  // 0

String str1 = “hi”;
String str2 = “hello”;

String str3 = str1 + str2; // hihello
```

3. Unary Operator

    - The unary operators require only one operand; they perform various operations such as incrementing/decrementing a value by one, negating an

expression, or inverting the value of a boolean.

Int I = -10;

Operator	Description
+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

- The prefix version (++result) evaluates to the incremented value, whereas the postfix version (result++) evaluates to the original value.
- If you are just performing a simple increment/decrement, it doesn't really matter which version you choose. But if you use this operator in part of a larger expression, the one that you choose may make a significant difference.

```
Int I = 10;
++I
I++

Boolean a = false;
Boolean b = !a;   //true
```

3. Equality, Relational, and Conditional Operators

The Equality and Relational Operators

- The equality and relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand.

```
==      equal to
!=      not equal to
>       greater than
>=      greater than or equal to
<       less than
<=      less than or equal to
```

```
Int a = 10;

Int b = 10;
```

If (a < b) {

}

The Conditional Operators

- The && and || operators perform Conditional-AND and Conditional-OR operations on two boolean expressions.

&& Conditional-AND
|| Conditional-OR

int a = 10;
int b = 1;
int c = 5;

if ((a < b) && (b < c)) {

}

4. The Type Comparison Operator instanceof

- The instanceof operator compares an object to a specified type.

Employee emp = new Employee();

if (emp instanceof Employee e) {

}

5. Bitwise and Bit Shift Operators

The bitwise & operator performs a bitwise AND operation.
The bitwise ^ operator performs a bitwise exclusive OR operation.
The bitwise | operator performs a bitwise inclusive OR operation.

Bitwise OR (|)

if either of the bits is 1, it gives 1, else it gives 0.

a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise OR Operation of 5 and 7
  0101
| 0111
_____
0111 = 7 (In decimal)

Bitwise AND (&)

if both bits are 1, it gives 1, else it gives 0.

a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise AND Operation of 5 and 7
  0101
& 0111

_____
 0101 = 5 (In decimal)

Bitwise XOR (^)

if corresponding bits are different, it gives 1, else it gives 0.

a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise XOR Operation of 5 and 7
  0101
^ 0111

_____
 0010 = 2 (In decimal)

Bitwise Complement (~)

with all bits inverted, which means it makes every 0 to 1, and every 1 to 0.

a = 5 = 0101 (In Binary)

Bitwise Complement Operation of 5

~ 0101

_____
 1010 = 10 (In decimal)

Statements
_____

1. Conditional Statement - execute the code based on some condition
    1. If condition
    2. If else condition
    3. If else if condition
    4. Switch case
2. Looping statement - execute the same code n number of times till the condition is false

1. For loop
2. While loop
3. Do while loop

n= 5

```
if (condition) {
        statement;
 }

If (condition) {
        statement;
 } else {
      Statement;
}

N =5

     *.
    **
   ***
  ****
 ******

Array
_____

Int a = 10

Int b = 20

Int c = 30

int[] array  = new int[3];

int array[]  = {10, 20, 30}
array[0]

int[ ][] array  =  new int[5][5];

array[0][1] = 10;
array[0][1] = 20;
array[0][1] = 30;

array[3] = 40; //ArrayIndexOutOfBoundException

sout(array[0])

int[ ][] array = {{1,2},
```

{2,3},
{3,4})

- Use to store the group of similar type of data in a single variable
- Size is fixed
- Array is index based
-

Types of Array
———————-

1. Single dimensional array
2. Multi dimensional array

0 1 2 3 4

0 1 2 3 4

0 1 2 3 4

0 1 2 3 4

0 1 2 3 4

OOPS:

Public class className{

// variables

Int I = 10

// constructor

// methods

}

1. Create a class Marks - getMarkDetails()-
2. Create a class students - getstudentdetails() - inside call marks and print the mark details
3. Create a class with main method and create the object for student and call the student details

Interface:

- Common functionality and different implementation
- Inside the interface we can create only abstract methods

- By default all the methods are public and abstract
- By default all the variables are public static and final
- We cannot create object
- We cannot create instance variable or constructor
- We can implement into another class

Abstract Class:
- Common functionality and common implementation

Encapsulation:

- Encapsulation is a technique that packages related data and behaviors into a single unit.
- Data Hiding or Information hiding
- Encapsulation is a technique for protecting data from misuse by the outside world

```
class Person {
    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public String getAge() {
        return age;
    }

public void setName(String name) {
    if (name == null || name.equals("")) {
        throw new IllegalArgumentException("name cannot be null or empty!");
    }

    this.name = name;
}

public void setAge(int age) {
    if (age >18 || age< 55) {
        throw IllegalArgumentException("Age must be from 18 to 55");
    }
    this.age = age;
}
}
```

Inheritance:

- Inheritance is the ability of a class inherits data and behaviors from another class.

- Note that only public and protected members of the superclass are inherited by the subclass.
- The subclass can freely add new members to extend features of the superclass.

superclass1 - gearBox()
superclass2 - gearBox()

SubClass extends Superclass1, superClass2

gearBox();

Single inheritance - one super and one sub class
Multiple inheritance - multiple super and one sub class
Multi level inheritance one super and child and child and child

Association
———————

- Relationship b/w 2 object we are calling as association
- Composition and Aggregation
- Composition
  - One Object owns another object
  - When the owner object destroy the dependent object also will destroy

```
Public class Employee{

// create the object for Address
// using address object we can call

}


Public class Address{

Public void getAddressDetails(){

}

}
```

- Aggregation
  - One object uses another object
  - When the owner destroy the dependent object will not destroy

Exception Handling
——————————

-Its an event, disturb the normal execution of the program

- Compile time exception (Checked Exception)
  - IOException
  - FileNotFoundException
- Runtime exception (Unchecked Exception)

```
String str = null;

str.toUpperCase()
```

```
int[] array= {10,20,30}

array[3];
```

Handling the exception
———————————

Try

Catch

Finally

Throw - to manually throw the exception

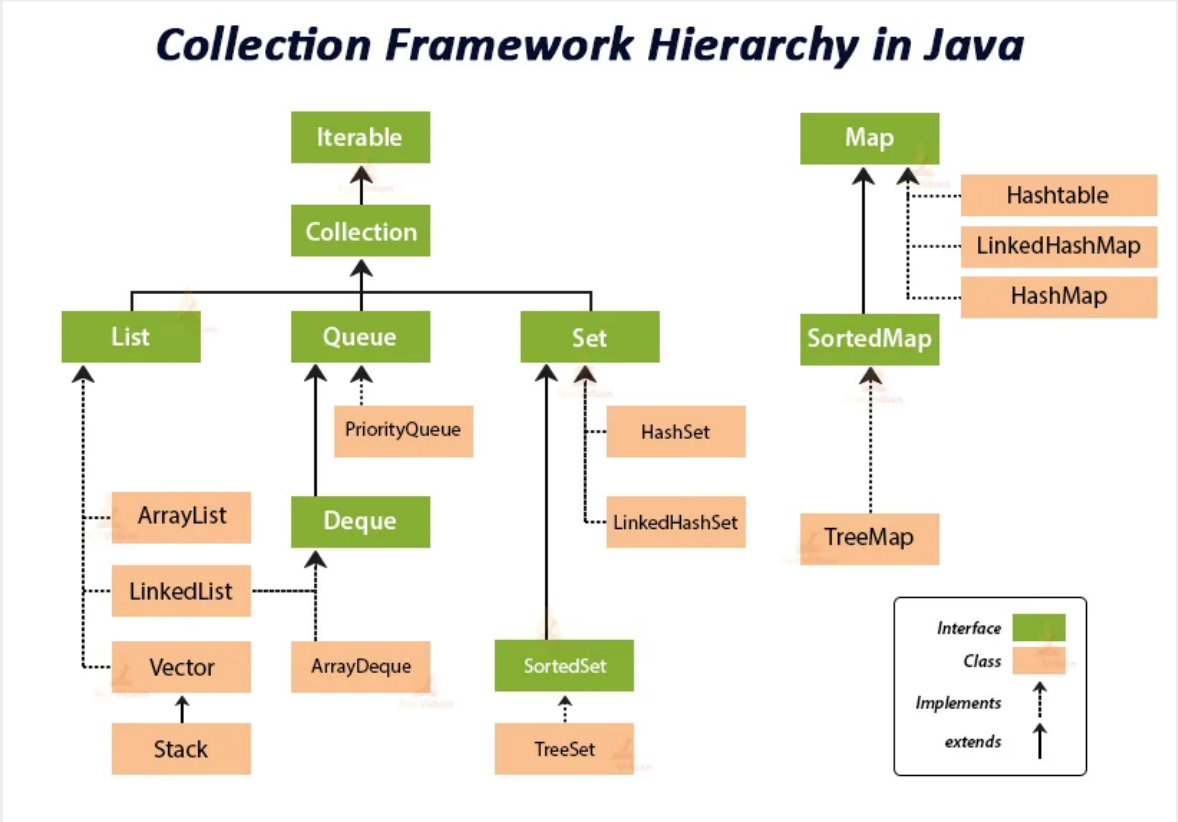
Throws - to throw the exception outside of the method

Application service. DAO

Try with resource

Collection
——————

- Its like a container
- It is use to store the similar type of objects
- We can store only object
- Size is dynamic



List

———

- Its an interface
- Implementations are ArrayList, LinkedList
- List allow the duplicate data
- Ordered collection
- List is mutable
- List is not Thread safe

T1. T2. T3

ArrayList:

————————

- Its similar to Array
- Its index based
- Size is dynamic
- Allocate the memory space for 10 elements
- The load factor is 0.75
- Ordered collection
- Will allow the duplicate data
- Mutable
- Not thread safe
- We cannot store the primitive datatype in collection
- We can store only object
- Performance is good for read operation

Syntax:

ArrayList<String> obj = new ArrayList<String>();

List<String> obj = new ArrayList<String>();

List<String> list = new ArrayList<>(); // from java 7, diamond operator

list.add(“Chennai”);
list.add(“Bangalore”);
list.add(“Kolkata”);
list.add(“mumbai”);
list.add(“Bangalore”);

list.get(0) ;//. Mumbai

LinkedList
————————

- DeQueue
- It will allow duplicate elements
- Size is dynamic
- Mutable
- Not thread safe
- Store only object, cannot store primitive data type
- Performance is poor for read operation
- Performance is good for insertion/deletion in the middle or beginning
- Ordered collection

Node

Data Next node address

Node

Pre node add. Data Next Node Add

5 10. 15 30

NULL	5	Next Node Add(10)

Prev Add (5)	10	Next Node Add(15)

Prev Node Add(10)	15	Next Node Add(30)

Prev Node Add(15)	30	Null

Syntax:

```
LinkedList<String> obj = new LinkedList<String>();

List<Employee> obj = new LinkedList<Employee>();

List<Integer> list = new LinkedList<>(); // from java 1.7, diamond operator

list.add(10);
list.add(20);
List.add(30);

List.get(0); // not valid for linked list
```

1. For each
2. Iterator
3. List Iterator
4. Lambda expression // recommendation
5. Sort by comparable
6. Sort by comparator

```
List list = new LinkedList();
List.add(10);

If (list != null  && list.size() > 0) {

        // for each
}

List<Integer> list = new LinkedList<>();

If (MapUtils.isNotEmpty(list)) {


}
```

1. student class/ Student Main
2. create list of student object
3. add all the student objects in Linked List
3. Sort based on Id. // comparable interface
4. Iterate the elements
5. Sort based on name // comparator using lambda

6. Iterate elements

Set
——

- It will store only unique data/elements
- Will not allow duplicate
- It will not override the exiting element
- Not an ordered collection
- Its mutable
- Not thread safe
- Interface

1. HashSet
2. LinkedHashSet - It will maintain the insertion order
3. SortedSet-> TreeSet - sorted

Assignment:
1. Create a TreeSet with String
2. Create a TreeSet with User defined Object

Map
——

- Store the data as a key value pair
- Key should be unique and value can be duplicate
- Allow only one null key and multiple null value
- Mutable
- Not thread safe
- Not an ordered collection
- Map is an interface

1. HashMap
2. LinkedHashMap - maintain the insertion order
3. SortedMap ——> TreeMap- sort ements based on the key

Accountno 1. —> Account 1. 16 0.75f. 50%

Accountno 2. --> Account 2

Account3

Fail fast

Fail safe

Queue

———————

- used to hold the elements about to be processed in FIFO(First In First Out) order
- Implementations are PriorityQueue and LinkedList

Some of the commonly used methods of the Queue interface are:

- **add()** - Inserts the specified element into the queue. If the task is successful, add() returns true, if not it throws an exception.
- **offer()** - Inserts the specified element into the queue. If the task is successful, offer() returns true, if not it returns false.
- **element()** - Returns the head of the queue. Throws an exception if the queue is empty.
- **peek()** - Returns the head of the queue. Returns null if the queue is empty.
- **remove()** - Returns and removes the head of the queue. Throws an exception if the queue is empty.
- **poll()** - Returns and removes the head of the queue. Returns null if the queue is empty.

```
Queue<String> queue = new PriorityQueue<>();
queue.add("Bangalore");

String element = queue.peek()
```

Serialization

———————-

- Conver Object to Byte

Deserialization

———————————

- Convert Byte to Object

- Implement the serializable interface
- Its a marker interface - Clonable, Serializable
-
-

Reflection

———————

Generics

———————

- Added java 1.5
- For type safety
- It will check type during compile time
- No need to type casting

```
List<String> list = new ArrayList<>();
List.add(1);
```

list.add(“Regu”);

(Int)List.get(0);

(String)list.get(1);

Concurrent Collections
———————————

1. CopyOnWriteArrayList
2. ConcurrentHashMap
3. SynchronizedList
4. SynchronizedSet
5. SynchronizedMap

CopyOnWriteArrayList<String> list = new CopyOnWriteArrayList<>();
List.add(“Bangalore”);

List<String> list = new ArrayList<>();
Collections.synchronizedList(list);

16 threads
Set<String> list1 = new HashSet<>();

Collections.synchronizedMap(list1);

ConcurrentHashMap<String, String> map = new ConcurrentHashMap<>(); 16
map.put(“city”,”Bangalore”);

MultiTreading
———————

1. Sequential Execution
2. Concurrent Execution

T1 - 10

T2. Shared resource - Synchronized

T3

1. Thread Class
2. Runnable interface

Application thread. Daemon thread

Thread Life cycle/States of Thread

1. Ready
2. Runnable - start()
3. Running - run()
4. Waiting
5. Dead

JVM Scheduler - part OS

Thread Priority - 5

10. 0

T1

T2

Obj1

Obj2

Preemptive Scheduler
Timeslicing

Sleep- halt the execution of the script and it will not release the lock, part of thread class
Wait - halt the execution of the script and release the lock, part of object class
Yield- part of thread

Thread.sleep(10000);

wait(10000);

Notify/notifyAll -part of object class

Synchronized
———————
Synchronised block
Synchronised method

Executor Framework
———————————
ThreadPool
———————
 - Group of worker thread

10 worker thread 10 normal thread

- Split the task and execution
- Runnable interface - run () - void()
- Callable interface - call() - Future

Executor - execute()- void, Runnable
ExecutorService - submit() - Future , Callable, Runnable
- invokeAll() - List<Future>

Assignment - 1. with submit()
2. With invokeAll()

Pending Topics
———————

Semaphore - done
thread starvation- done
thread local - done
deadlock - done
race condition,
fork join pool - Done
read write lock - done
Phaser

Thread 1.

Thread 2

Thread1

```
synchronized(String.class){

        synchronized(Integer.class){
        }
}
```

Thread 2

```
synchronized(String.class){

        synchronized(Integer.class){
        }
}
```

Kill -3

JViwer

T1 T2 T3

Deadlock
————————-

Thread1. Thread2

Thread1
————————

```
synchronized(String.class){
        synchronized(Integer.class){
        }
}
```

Thread2
————————
```
synchronized(Integer.class) {
        synchronized(String.class){
        }
}
```

Object Lifecycle
————————

In Java, it has seven states in Object lifecycle. They are,

1. Created
2. In use
3. Invisible
4. Unreachable
5. Collected
6. Finalized
7. De-allocated

Employee emp= new Employee();

Created
- New memory is allocated for an object.
- Assigned to some variable and then it directly moves to the In Use state.

In use
- one strong reference are considered to be “In Use”.

Invisible
- no longer any strong references that are accessible to the program

Unreachable
- An object enters an “unreachable” state when no more strong references to it exist.

Collected
- when the garbage collector has recognized an object as unreachable and readies it for final processing as a precursor to de-allocation.

Finalized
- An object is in the “finalized” state if it is still unreachable after it’s finalize method,
-

De-allocated
The de-allocated state is the final step in garbage collection.

Thread.yield