

JAVA:

1. Java is a high-level programming language.
2. Java is a simple programming language where writing, compiling and debugging is made easier.
3. It helps to create reusable code.

Features of java (why we go for java)

1. Platform independent.
2. Open source
3. Multithreading.
4. More secure.
5. Portable.

Platform independent.

During compilation Java code converts into byte code, this byte code can be used to run in any platform like windows, macOS, Linux etc. that supports java.

Open source

A program in which source code is available to the general public for use and/or modification from its original design at free of cost is called open source.

Multithreading

Multithreading enables the program to perform several tasks simultaneously.

More secure

Java provides firewall between the computer and the application. So, doesn't grant unauthorized access. It takes place by the interface called Serialization.

Portable

"Write Once Run Anywhere".

Java code written in one machine can run on another machine.

Components / Terminologies/ Elements of Java

JDK is Java Development Kit is essential whenever we need to write, compile and run a program in Java. It has JRE and development tools.

JRE is Java Runtime Environment which contains predefined classes(library files) and JVM

JVM is Java Virtual Machine which is used for memory allocation and object creation.

Features of Java 1.7

- We can use String in switch case
- Underscores Between Digits in Numeric Literals.
- Automatic Generic conversion
- Handle multiple Exception in single catch block

OOPS:

OOPS is Object Oriented Programming Structure. It's a method of implementation in which program is organized as collection of Class, Method, Object.

Class is the collection of methods and objects.

Methods is nothing but set of actions to be performed. It is the block of code that runs only when it is called. We can implement our business logic in methods.

Object is the instance of the class. By using object only, we can call the methods in the class. It is a runtime memory allocation

Principles of OOPS

1. *Inheritance*
2. *Polymorphism*
3. *Abstraction*
4. *Encapsulation*

ENCAPSULATION:

- ✚ It is a structure of creating folders.
- ✚ It wraps the data and code acting on a data together in to a single unit.
- ✚ Example of encapsulation is POJO class.
- ✚ It is otherwise called data hiding.

INHERITANCE:

Accessing one class properties in other class by using extends keyword without multiple object creation.

Advantage:

1. We can avoid code redundancy(repetition).
2. Reduce memory wastage.
3. Used for reusable code purpose.

Types of Inheritance;

1. *Single Inheritance*
2. *Multilevel Inheritance*
3. *Multiple Inheritance*
4. *Hybrid Inheritance*
5. *Hierarchical Inheritance*

1 Single Inheritance:

Combination of one parent class and one child class (or) One parent class supporting into one child class. (or) One child class acquires the properties of one parent class.

2 Multilevel Inheritance:

More than two parent class supporting into one child class in tree level structure. (or) One child class acquires the properties of more than one parent class sequentially or in the chain structure.

3 Hierarchical Inheritance:

One parent class supporting into more than one child class. (or) More than one child class acquires the properties of the single parent class.

4 Multiple Inheritance:

More than two parent class supporting into one child class parallelly at a time. (or) One child class acquires the properties of more than one parent class simultaneously.

But Multiple inheritance is not supported in Java because of

1. *Priority problem/dead lock problem*-when multiple parent methods have same name and arguments, compiler will not know which method should be called first.
2. *Compilation error/syntax error*-after the extends keyword we can mention only one class name.

We can achieve it in java through interface concept

5 Hybrid Inheritance:

It is the combination of single and multiple inheritance.

POLYMORPHISM

One activity that can be performed in different ways. (or) One method that can be executed in many ways is called polymorphism.

Two types of polymorphism

1. Method overloading (Static Binding /Compile time polymorphism)
2. Method overriding (Dynamic Binding/ Run time polymorphism)

1 Method overloading/ Static Binding

When we are going to overload a method again and again with same method name by differing its argument like data type, data type count, data type order.

2 Method overriding/ Dynamic Binding

When we are not satisfied with the parent class method, we can create the same method (with exact same method name) in our child class and we can write our required business logic.







ABSTRACTION

Abstraction is a process of hiding the implementation part and shows only its functionalities.

We can achieve abstraction in two ways,

-  Interface
-  Abstract class

1 Abstract class

-  Class which is declared as abstract is known as abstract class. We can achieve partial abstraction using Abstract class.
-  It can have both abstract method and non-abstract method.
-  It can have constructor and static method also.
-  We can't create business logic in abstract method.
-  Abstract class cannot be instantiated.
-  To implement business logic in Abstract class we have to inherit it to other class by using extends keyword and then we can provide implementation to the abstract methods in it.

2 Interface

- ✚ Interface in java is used to achieve full abstraction in Java.
- ✚ Instead of class we have to use interface keyword.
- ✚ Interface supports only abstract methods.
- ✚ We can't create any object for interface.
- ✚ public abstract is default in interface methods.
- ✚ To implement business logic in interface we have to inherit it from other class by using implements keyword and then we can provide implementation to the abstract methods in it.

DATA TYPES:

Data type specifies that different type of size and type of value stored in the variables.

VARIABLES

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type. Variable is a name of memory location.

Types of Variables

- Local variable
- Global variable/instance variable
- Static variable/class variable

1. Local variable:

1. Variable will be visible only inside the block or method (we will declare inside the block or method).
2. Whenever we declare we must initialise some value, otherwise it will get assigned with garbage value.
3. we cannot give access specifiers. (remove invalid modifiers).
4. Local variable is activated when method created and deactivated when method is destroyed.
5. Stores in Stack memory.

2. Instance (Object)/Global variable:

1. Variable will be visible all over the class (You will declare outside all methods but inside class).
2. we don't need to initialise a value while declaration, it will take default value.
3. we can give access specifiers.
4. Global variable is activated when object created and deactivated when object is destroyed.
5. Stores in Heap memory.

3. Class/Static variable:

1. Global Variable with static keyword.
2. When a variable declared as static, then a single copy of variable is created and shared among all object at class level.
3. Static variable is activated when class created and deactivated when class is destroyed.
4. Stores in stack memory.

Wrapper class:

1. Classes of datatypes is called wrapper class.
2. It is used to convert any datatype into object.
3. All classes and wrapper classes default value is null.

7. STRING:

- Collection of characters or words enclosed within double quotes is called string.
- String is a class in Java.
- String is index based.
- Eg.” greens technology”.

Basic Methods:

charAt()	To find the character at the given index
length()	To find the length of the string
toUpperCase()	Convert given string into uppercase
toLowerCase()	Convert given string into lower case
equals()	To check whether two strings are equal (case sensitive)
equalsIgnoreCase()	To check whether two strings are equal without considering upper or lower case
startsWith()	To check prefix of the string
endsWith()	To check suffix of the string
replace()	To replace particular character by other
indexOf()	Find the index of the given character (if two char is present considers the first)
lastIndexOf()	Find the index of the character (if two char is present considers the last)
contains()	To check whether the string contains given sequence
split()	Splits from the given character and delete that character
substring()	Print string from particular given index to particular given index
trim()	To remove only prefix and suffix spaces
isEmpty()	To check whether the string is empty or not
concat()	To combine two strings in immutable strings
append()	To combine two strings in mutable strings
identityHashCode()	To find the address of datatype.

String types:

1. Literal String:
2. Non-Literal String:

1. Literal String:

String s= "Hello";

In case of duplicates, it allocates same memory. Stores inside the heap memory i.e., string pool constant.

2. Non-Literal String:

String s=new String("Hello");

In case of duplicates, it allocates different memory. Stores in the empty space of the heap memory

Immutable string:(same as literal string)

- ✚ In case of duplicates it allocates same memory.
- ✚ We can't change value in the memory.
- ✚ While combining two strings, combined string will be stored in different memory.
- ✚ When we combine two immutable string, we can use *concat()*

Mutable String:(same as Non literal string)

- ✚ In case of duplicates it allocates different memory.
- ✚ We can change value in the memory.
- ✚ While combining two strings, combined string will be stored in the first-string memory.
- ✚ When we combine two mutable string, we can use *append()*

String buffer

StringBuffer s=new StringBuffer("Hello");

- ✚ It is synchronized.
- ✚ Thread safe.
- ✚ Multiple threads can't access simultaneously.

String builder

StringBuilder s=new StringBuilder("Hello");

- ✚ It is asynchronous.
- ✚ Non thread safe.
- ✚ Multiple thread can access simultaneously.

Why String is immutable in java?

Whenever we are declaring any String it will be stored in String pool constant which makes that we cannot change value in the particular memory.

ARRAYS:

- ✚ In a single reference mean we can store multiple value of similar data types.
- ✚ Array is an index based one.
- ✚ Array allows duplicates.
- ✚ Array is a fixed memory allocation or static memory allocation.
- ✚ We can access all the array elements by for loop or Enhanced for loop
- ✚ We can initialize array in two ways
 - *Datatype[] refName = new Datatype[size];*
 - *Datatype[] refname = { value1,value2,....};*

Two-dimensional array:

It is matrix format structure where array will be organized in rows and columns.

- *Datatype[] [] refName = new Datatype[row size][column size];*

Advantage of array:

1. In a single reference name we can store multiple values.

Disadvantage of array:

1. It supports only similar data types.
2. Array is static or fixed memory allocation.
3. Memory wastage will be very high. Because we are allocating memory in compile time.
4. We cannot change memory size after array initialization.

To overcome these disadvantages we go for collection.

COLLECTION:(Interface)

- ✚ Storing multiple values of dissimilar datatypes in a single variable.
- ✚ It supports dissimilar datatype (different datatypes).
- ✚ No memory waste compared to array.
- ✚ Dynamic memory allocation/run time memory allocation.
- ✚ We can't instantiate (create object) object for interface.

Interfaces in Collections:

1. List → Interface
2. Set → Interface
3. Map----doesn't come under collection; it is a separate interface in java

1 List:

- ✚ List is an index based one.
- ✚ List allows duplicates.
- ✚ List prints the value in insertion order
- ✚ Iterate using normal and enhanced for loop.

Classes of list interface:

1. **ArrayList**
2. **LinkedList**
3. **Vector**

ArrayList	LinkedList
Insertion, deletion -performance wise slow	Insertion, deletion -performance wise fast
Searching, retriving is fast	Searching, retrieving is slow

ArrayList	Vector
Asynchronised	Synchronised
Not thread safe	Thread safe

METHODS	
add(elements)	To add element into the list
size()	To get number of elements in the list
get(index)	To get the value of the given index
indexOf(element)	To return(1 st occurance) the index of the given elment
lastIndexOf(element)	To return(last occurance) the index of the given elment
add(index,element)	To add the element into the list in the particular index
set(index,element)	To replace the element in particular index
remove(index)	To remove the given index
isEmpty()	To check the list is empty or not
list1.retainAll(list2)	It retains only the common element in the destination list. Left side list
list1.removeAll(list2)	It removes the common element in the destination list.
contains(element)	Used to check the given element is present in the list or not
addAll()	To add all the elements from one collection to another

2 Set

- ✚ Set interface extends collection interface.
- ✚ Set is not index based; Set is a value based
- ✚ Since it is not index based, index-based methods in list will not be allowed here.
- ✚ Set won't allow duplicates
- ✚ To iterate all the elements from Set we can use only enhanced for loop.
- ✚ Set allows only one null.

Types of Set

1. **HashSet**- It will print in random order.
2. **LinkedHashSet**- It will print in insertion order.
3. **TreeSet**- It will print in ascending order.

Methods supported by list not by set:

- ✚ get();
- ✚ indexOf();
- ✚ lastIndexOf();
- ✚ add(index,value);
- ✚ set(index,value);

we can convert set into list by addAll(); method. And it won't allow duplicate.

3 Map

- Map is a (key, value) pair combination.
- Key does not allow duplicates. Value allows duplicates.
- We cannot iterate map directly. After using `entrySet()` method we can iterate map entries.

classes implementing map:

- `HashMap` → random order(unordered)
- `LinkedHashMap` → insertion order
- `TreeMap` → ascending order
- `HashTable` → random order(un ordered)

	Key	Value
HashMap	1 null	'n' null
LinkedHashMap	1 null	'n' null
Treemap	ignore null	'n' null
HashTable	ignore null	ignore null

HashMap	HashTable
<ul style="list-style-type: none">Asynchronized (parallel access)Not thread safeAllows null values	<ul style="list-style-type: none">Synchronized(sequential access)Thread safeIt wont allow null for key and value.

Methods in map:

method	
<code>mp.put(key,value)</code>	To add elements in the map
<code>mp.get(key)</code>	Print values based on key
<code>mp.containsKey(key)</code>	To check given key is present or not
<code>mp.containsValue(value)</code>	To check given value is present or not
<code>mp.keySet</code>	It is used to separate key from value
<code>mp.value</code>	It is used to separate value from key
<code>mp.entrySet</code>	It is used to iterate the map
<code>entry.getKey()</code>	To print key in iteration
<code>entry.getValue()</code>	To print values in iteration

- In map we cannot iterate the values directly using normal for loop/ enhanced for loop.
- We can iterate through `entrySet()` method because `entrySet()` method is a method which is going to convert each and every key and value by combination into Entry.
- So that we are converting two different elements into a single element.

Entry is an Interface.

List	Set
<ul style="list-style-type: none"> • It is a Index based one. • It prints in insertion order. • It allows duplicates. 	<ul style="list-style-type: none"> • It is a value based one. • It prints in random order. • It won't allow duplicates.
Set	Map
<ul style="list-style-type: none"> • It is a value based one. • It prints in random order. • It won't allow duplicates. 	<ul style="list-style-type: none"> • It is key and value pair. • Here key+value is one entry. • Key ignore the duplicate value and value allow the duplicates.

Collections:

Collections is a utility class in which we have lots of predefined methods which we can apply over collection objects.

Eg: Collections.min(), Collections.max(), Collections.sort().

CONSTRUCTOR:

A special method which will get automatically invoked when we create object for the class (implicit call).

Class name and constructor name should be same

Constructor doesn't have return type.

It supports method overloading but won't support method overriding.

Purpose of the constructor is to initialize values to the variables.

Types of Constructor:

- **Parameterized constructor/** argument-based constructor.

In parameterized constructor we have to pass the argument in object

- **Non parameterized constructor/** non-argument-based constructor.

In non-parameterized constructor we don't want to pass the argument in object

Return type not allowed in constructor:

- Constructor is not directly called by your code; it is called by memory allocation and object initialisation in the run time.
- Its return value is opaque to the user so we can't mention it.

Default constructor:

If we didn't create a constructor explicitly it will create a constructor by default which will not be visible.

Constructor	Method
Special method, whenever object is created it will get invoked automatically	We need to call the method using object
Implicit – method calling	Explicit – method calling
Constructor name and Class name should be same	We can give any name
Constructor won't have any return type	Method always have return type

Constructor Chain

- ✚ The process of calling one constructor from another constructor with respect to current object is called constructor chaining.
- ✚ By using this() and super() methods we can achieve constructor chaining.

this() and **super()** method should be always given in the first line of the constructor.

this → represents the current class. is used to call class level variables and methods.

this() → represents the current class constructor. is used to call class level constructor.

super → represents the parent class. is used to call parent class level variables and methods.

super() → represents the parent class constructor. is used to call the parent class constructor.

EXCEPTION HANDLING:

Exception

Exception is an unexpected event when it takes place program will be terminated abnormally. To avoid the abnormal termination, we can use exception handling mechanism.

Types of exceptions:

1. Unchecked (runtime exception)
2. Checked (compile time exception)

Exception is the super class of all checked (compile time) and unchecked(runtime) Exceptions.

Throwable is the super class of exception and error.

Exception	Error
Exceptions are the conditions that occur at runtime/compile time and may cause the abnormal termination of program.	Errors are the conditions that occur at runtime and may cause abnormal termination
Exceptions are divided into two categories: <ol style="list-style-type: none"> 1. checked 2. unchecked 	Errors belong to unchecked type and mostly occur at runtime.
It is mostly caused by the program written by the programmer.	Errors are mostly caused by the environment in which application is running.
It is recoverable	It is irrecoverable
NullPointerException SQLException	OutOfMemoryError IOException StackOverflowError

Unchecked exception: Whenever the exception occurs at runtime is called runtime exception.

- `ArrayIndexOutOfBoundsException`
- `IndexOutOfBoundsException`
- `StringIndexOutOfBoundsException`
- `NoSuchElementException`
- `SessionNotCreatedException`
- `IllegalStateException`

Checked exception: Whenever the exception occurs at compile time is called compile time exception.

- `FileNotFoundException`
- `AwtException`
- `InterruptedException`
- `IOException`
- `SQLException`
- `ClassNotFoundException`
-

Exception Handling

try - Code which is expected to throw exception will be kept inside try block.

catch - Catch block acts as a solution block, It will Handle the exception.

finally - Whether exception occurs or not, handles or not, finally block always be executed.

throw

- Using throw keyword, we can explicitly throw an exception.
- Using throw, we can throw only one exception at a time.
- Throw keyword is used inside a try block

throws

- Using throws keyword, we can handle the exception at compile time.
- We can declare more than one exception at a time.
- Throws is given in method signature.

User defined Exception

To customise your own Exception

- Create a class → Class name → Exception name. Eg: `InsufficientFundException`
- Your class should extend Exception class → Eg: `EmployeeNotFoundException` extends `Exception`
- Override `getMessage()` method from `Throwable` class --- To customise message for your customised Exception
- throw your Exception → Eg: `throw new ExceptionName();`
- You can use throws to declare the Exception in method signature or u can handle it using try and catch block!

ACCESS SPECIFIER:

Scope:

Defines the level of access for a variable, method and class.

Access specifier

1. **Public** – It's a global level Access Specifier. When we use public as access Specifier, we can access in same package as well as different package by using object and extends.
2. **Protected** – When we use Protected as Access Specifier, we can access by using object and extends in the same package and we can access using extends in different package.
3. **Default** – It's a package level Access Specifier. We can access by using object and extends when it comes to same package.
4. **Private** – It's a class level Access Specifier. We can use it only in a class.

ACCESS MODIFIERS

Change or modify the accessibility of the variable, class and method.

Static – It can be declared at method and variable level only. If we declare as Static, we can call using method name and variable name without creating object.

Abstract – It can be declared at class and method level only. If we declare class as abstract, we can't create object and if we declare method as abstract, we can't write business logics.

Final – It can be declared at class, method and variable level. If we declare class as final, we can't inherit. If we declare method as final, we can't override. If we declare variable as final, we can't change the value.