DAY-1

1. Features of Java

Object-Oriented:- This makes java applications easy to develop and maintain, compared to structured programming language.

Portable and Platform Independent:- Java source code is compiled and
converted into bytecode. this bytecode can run on multiple platforms i.e. Write Once and Run Anywhere(WORA), we can compile the java code in one Operating System and execute it on another Operating System.

Robust:- Robust means strong. Java is having a very good memory management system in the form of a heap memory management system, it is a dynamic memory management system, it allocates and deallocates memory for the objects at runtime.

- JAVA is having very good Exception Handling mechanisms, because, Java has provided a very good predefined library to represent and handle almost all the frequently generated exceptions in java applications.

Multithreaded:- Java supports multithreading to enhance performance.
Secure:- JAVA has provided an implicit component inside JVM in the form of a
“Security Manager” to provides security against malicious code. Java has provided very good predefined implementations for almost all well-known network security.\

LIST 1

- **1. private: accessible in the same file**
- **2. default: accessible in the same package**
- **3. protected: accessible in the same package; accessible in all packages w/ "extend" E.g. otherpackage class extend package1 class**
- **4. public: accessible in all packages**

Day-2

Identifiers : Identifiers is a name assigned to programming elements like variable, method name.

Rult :

- Identifiers can start with a _ or $ or alphabet.
- Should not start with number.

Literals :
Integer literal : byte, short, int, long.
Floating point literal : float, double.
Boolean literal : boolean.
String literal : String.

A. Primitive Data types :

1. Numeric Data Types (1 byte = 8 bits)

a. Integral data types/ Integer Data types:
byte ------> 1 bytes ----> 0

short------> 2 bytes-----> 0
int--------> 4 bytes-----> 0
long-------> 8 bytes-----> 0
b. Real Number Data Types:
float------> 4 bytes----> 0.0f
double-----> 8 bytes----> 0.0

2. Non-Numeric Data types:

char ---------> 2 bytes---> ' ' [single space]
boolean-------> 1 bit-----> false

Primitive Data Types : Wrapper Classes (it is a class representation of wrapper class)

byte	: java.lang.Byte
short	: java.lang.Short
int	: java.lang.Integer
long	: java.lang.Long
float	: java.lang.Float
double	: java.lang.Double
char	: java.lang.Character
boolean	: java.lang.Boolean

System.out.println(Byte.MIN_VALUE+"----->"+Byte.MAX_VALUE);

Type Casting

1. Implicit Type Casting
2. Explicit Type Casting

Implicit Type Casting : from lower data type to higher data type (no type casting required)
Explicit Type Casting : from higher data type to lower data type (type casting required)

Type Checking is the responsibility of compiler and Type Casting is the responsibility of JVM.

DAY - 3

The benefits of Object-oriented programming:-

- Reduce complexity
- Easier maintenance
- Code reusability
- Faster Development

Difference between Class and Object:

Class is a template or a blueprint for the objects. The main purpose of the class is to represent all real-world entities in Java applications.
whereas **Object** is an instance of the Class.

Instance variable in Java

A variable that is created inside the class but outside the method is known as an instance variable. The instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

1. static elements (instance variables and methods)
2. non-static elements(instance variables and methods)

Static elements

All static elements will be loaded into the RAM first and other non-static variables will be in hard-disc in the form of byte-code in dot class file.

When we execute the program the JVM searches for the main method and executes it. Since static elements are loaded into the RAM, they will be available for the CPU for the execution, but non-static members will not be loaded into the RAM initially. so they will not be available to the CPU for the execution.

We can not access the non-static elements from the static area directly.

By using some procedure if we can transfer the content of the hard disk to RAM (loading the non-static elements into the RAM) so that x will be loaded in the RAM and it will be available to the CPU for execution.
The procedure of loading the contents of the hard disk to the RAM dynamically at Runtime is done by creating an object of a class, this is the need for creating an object.

‘new’ operator:

An important responsibility of the new operator is to transfer the contents of the .class file from the hard disk to the RAM.

- As soon as the 'new' operator is encountered by the JVM, JVM will allocate some memory space for all the non-static members present inside the dot class file and it will load all the non-static elements from the hard disk to the RAM.
- initializes with their default values if the value is not given, and if any non-static will be there then the signature and address of the method will be loaded, not the body of the method.
- And then the address of this memory location will be assigned to the variable “d1”, where d1 is a variable of class Demo type.

Note: the memory in the RAM which is reserved for the non-static elements of the dot class file is known as Object in java.

- Now we can say that “obj” is the variable of the class Main type which contains the address of the object.
- Now if we change the value of the x through the “obj” variable, like obj.x=20; then the change will be done only inside the RAM, but the value in the dot class file which is in the hard disk will not be affected.
- Since “obj” points to the object of class Main, it is not an object, it is only a reference to an object. After execution of the main method, the main method will also be deleted from the RAM and the reference variable created in the main method “obj” will also be deleted and the address to the object pointed out by “obj” will also be lost.
- the object which has no variables to point them treated like garbage. In Java, it is the duty of the garbage collector to clear this garbage and free the memory in the RAM.

Encapsulation

This is the basic principle of object-oriented programming. According to this principle, we bundle the data and methods that operate on that data in a single unit.

static elements - Only one copy of that member would be shared by all objects.
static members belong to the class, whereas non-static members belong to the object.
non-static elements - specific for object

Static method - The Common functionality of all objects in the application must be defined as static.
Non-static - functionality belongs to a particular object.

Day - 4

Int java Object class is the parent class for all objects

Methods in Java

- A Method in java is a set of instructions, it will represent a particular action in java applications.
- A method is a block of code that only runs when it is called.

Note: if a method takes a concrete class as a parameter, then in order to call that method, we can pass following 3 things:

- same class object
- its child class object //we will talk about the child classes in upcoming session
- null

Polymorphism:

Defining more than one functionality with the same name in the same class is known as polymorphism.
The main advantage of Polymorphism is "Flexibility".

Static polymorphism: If the Polymorphism is existed at compilation time then it is called Static Polymorphism.
example: method overloading (same method name, but the parameter will be different)

- It is also known as compile time polymorphism, i.e. which method will be called, decided at compile time only.

Dynamic Polymorphism: If the Polymorphism is existed at runtime then that Polymorphism is called Dynamic Polymorphism. example: method overriding:-
(same method name ,and same parameter)
Method overriding we achieve through inheritance.
It is also known as runtime polymorphism, i.e. which method will be called , decided at runtime.

Constructor:

It is used for initialisation of the variables.

Note:- we can have a dot java file of a class, without a constructor, but we can't have a dot class file of a class without a constructor.
At least the default constructor must be there inside the dot class file of a class.

- A method can be static, where the static keyword is not applicable with a constructor.
- A method can be abstract where as abstract keyword is not applicable with the constructor.
- Final keyword is applicable with the method, whereas it is not applicable with the constructor.

We know static keywords belong to a class rather than the object of a class. A constructor is called when an object of a class is created, so no use of the static constructor.

If we are declaring a constructor as abstract as we have to implement it in a child class, but we know a constructor is called implicitly when the new keyword is used so it can't lack a body

One of the important properties of a Java constructor is that it can not be final. As we know, constructors are not inherited in java. Therefore, constructors are not subject to hiding or overriding

Note: if we keep any constructor in our dot java file, then java compiler won't provide a default constructor to our dot class file.

What is ‘this’ keyword ?

‘this’ keyword points to the current object.

- Points to the current class object.
- Differentiate between local & instance variables.
- Calling a constructor from another constructor of the same class.

Note: ‘this’ keyword we can not use inside the static area.

Since the static methods doesn't have (belong to) any instance you cannot use the "this" reference within a static method.

Calling a constructor explicitly:

A constructor will be called automatically whenever we create an object of a class.

1. From the another constructor of the same class (using 'this' keyword)
2. From the constructor of its child class. (using 'super' keyword)

Note:- if we call a constructor explicitly from the another constructor then that call must be the first line.

Pure encapsulation:

This concept says that, we need to mark our data as private and expose those data through the public methods (like getter and setters methods).

Java Bean class aka POJO (plain old java object):

It is a reusable, purely encapsulated java class which should have following properties:

1. The class must be public.
2. All the fields should be private.
3. For each field there should be corresponding public getter and setter methods.
4. It should have a zero argument constructor.
5. It may have a parameterized constructor (it is not the minimum requirement).

DAY-5

String class

String is an object that represents a sequence of characters. There is a predefined class in java called String.

By literal
String s = “message”

In Java, the **JVM** maintains a string pool to store all of its strings inside the memory. The string pool helps in reusing the strings.
Here, we are directly providing the value “Welcome”. Hence, the compiler first checks the string pool to see if this string already exists.

- If this string already exists, the new string is not created. Instead, the new reference points to the already existing string “Welcome”.
- If the string doesn't exist, the new string “Welcome” is created inside this string pool area.

By new keyword:

This will create a new string in the heap memory, even if the string exists in the string pool.

Java Strings are Immutable

Once we create a String we cannot change it.

Why String objects are immutable in Java?

As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Masai". If one reference variable changes the value of the object, it will be affected by all the
reference variables. That is why String objects are immutable in Java.
Note: If we want to create a mutable (modifiable) String object, we should use either StringBuffer or StringBuilder classes. both classes belong to java.lang package also.

String	**:** waste memory, thread safe & good performance.
StringBuilder	**:** good memory, not thread safe & good performance.
StringBuffer	**:** good memory, thread safe & poor performance.

String compare:

equals() : equals() method compares the original content of the string.

Using == operator : The == operator compares references not values.

compareTo() : compareTo() method compares values lexicographically and returns an integer value.

Array

Note: at the time of creating an array object, all the variables inside the array will be initialized with their default value.

If we try to access an element from an array beyond its size, then it will raise a runtime exception called **ArrayOutOfBoundException**.

Command Line Argument:

The command-line arguments in Java allow the programmers to pass the arguments during the execution of a program. The users can pass the arguments during the execution by passing the command-line arguments inside the main() method.

DAY-6

Inheritance

Inheritance is one of the most important features of OOP(Object-Oriented Programming).

Inheritance is a relationship[Parent-Child] between classes, it will bring variables and methods from one class[Super class / Base Class / Parent Class] to another class[Subclass / Derived Class / Child Class] in order to reuse.

The main advantage of Inheritance in Java is:
1. Code Reusability
2. To get Runtime polymorphism (using method overriding)

- **Interface** is defined when you don't have implementation details.
- A **concrete class** is defined when you have complete details about the object.
- **Abstract class** is like a transition(transform) between interface and concrete class.

Types of Inheritance:
1. Single Inheritance - ALLOWED
2. Multilevel Inheritance - ALLOWED
3. Hierarchical Inheritance - ALLOWED
4. Multiple Inheritance - NOT ALLOWED
5. Hybrid Inheritance- NOT ALLOWED

Multiple inheritance & Hybrid inheritance has diamond problem.

Diamond problem : when a method is inherited from two classes JVM will not know which method to override. Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes.

In java programming, multiple and hybrid inheritance is supported through interface only.

Method Overriding

As we know that object of a child's class can access the method of its parent class also. but, if the child class object is not satisfied with the implementation of the inherited method, the child class can re-implement the inherited method with his implementation, this concept is known as Method Overriding in java.

- Method overriding is used for runtime polymorphism

Note: We can not override a static method :

Can we overload a static method?

The answer is Yes. We can overload static methods. But remember that the method signature must be different. For example, consider the following Java program.

Can we overload if the method signature differs only by static keyword?

The answer is **No.** We cannot override two methods if they differ only by static keyword.

Valid overload methods

```
public static void Run1() {
   System.out.println("SuperBike");
}
public void Run1(int x) {
   System.out.println("SuperBike");
}
```

Can we override a static method?

No, we cannot override static methods because method overriding is based on dynamic binding at runtime and the static methods are bonded using static binding at compile time. So, we cannot override static methods.

- If we call a static method by using the parent class object, the original static method will be called from the parent class.
- If we call a static method by using the child class object, the static method of the child class will be called.

Binding is a mechanism creating a link between method call and method actual implementation.

Static (or) early binding :

- Happens at compile time
- Actual object is not used
- Ex. Method overloading

Dynamic (or) Late binding :

- Happens at run time
- Actual object is used
- Ex. Method overriding

super keyword

The super keyword in Java is a reference variable that is used to refer to immediate parent class object.

Usage of Java super Keyword
1. super can be used to refer to the immediate parent class instance variable.
2. super can be used to invoke the immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Dynamic or Runtime polymorphism:

It is a process in which a call to an overridden method is resolved at runtime rather than Compile-time.

Upcasting: A a=new B();   a.run()//upcasting // B class method will be called in runtime.

Note: Runtime polymorphism can't be achieved by data members.

```
class Bike{
        int speedlimit=90;
}
class Honda3 extends Bike{
        int speedlimit=150;
        public static void main(String args[]){
                Bike obj=new Honda3();
                System.out.println(obj.speedlimit);  //90  -ans
}
```

instanceof operator:

The java instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or an interface). **It also return true if it is a sub class.**

```
class Animal  {
        public static void main(String args[]){
                Animal a=new Animal();
                System.out.println(a instanceof Animal);//true
        }
}
```

Downcasting

As we know that to a parent class variable we can assign the child class object also, and from that parent class variable if we try to call any overriden method then due to Runtime polymorphism the overriden method will be called. but if a parent class reference points to a child class object, with that parent class refernce we can not call the child class specific methods, which are not available inside the parent class.
to call the child class specific method from the parent class reference variable we need

to downcast the parent class variable to the appropriate child class object.

Note: We can downcast the parent class variable to the child class object only if the Parent class variable points to the Child class object , otherwise it will throw a runtime exception called ClassCastException.

Animal parent = new Dog();
Dog d = (Dog) parent;

Below code provided class cast exception
```
Vehicle v = new Vehicle();
Bike b = (Bike) v;
```

Final Keyword

The final keyword in java is used to restrict the user.

If you make any variable final, you cannot change the value of a final variable

If you make any method final, you cannot override it inside the child class.

If you make any class as final, you cannot extend it. final class does not have the child class.

Object class

It is present in java.lang package. It is the super class for all classes in java by defaut.

DAY-7

Packages

Package in Java is a mechanism to encapsulate a group of classes, sub-packages and interfaces.

- Preventing naming conflicts.
- Group the related classes, interfaces, etc logically.
- Providing controlled access.

System (java.lang) . out (static member of the System class) . println (overloaded method of printstream)

Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- Abstract class (0 to 100%)
- Interface (100%) Since Java 8, we have default methods.

Abstract class

- It **can have abstract** and **non-abstract methods**.
- It cannot be instantiated (object creation).
- It **can have constructors** and **static methods** also.
- It **can have final methods** which will force the subclass not to change the body of the method.

Rules:
1. If there is an abstract method in a class, that class must be abstract.
2. If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.

DAY-8

Interface

An interface in Java is a blueprint of a class.
The interface in Java is a mechanism to achieve 100% abstraction.
Since Java 8, we can have default and static methods in an interface.

Note :
We use static methods in the interface because it cannot be overridden. It is for constant methods for all implementations.

Defaut method to allow the developers to add new methods to the interfaces without affecting the classes

Interface can have variables. Even if we dint mention anything? It is by default
public static final int number=10;

- Interface variables are static because java interfaces cannot be instantiated on their own.
- The final modifier ensures the value assigned to the interface variable is constant

Marker or tagged interface in java:

An interface that has no member is known as a marker or tagged interface, for **example, Serializable, Cloneable** etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operations.

Early and Late Binding in java

Connecting a method call to the method body is known as binding.

When the type of an object is determined at compile time it is known as **early Binding.**
Note : If there is any private, final or static method in a class, there is static binding.
Dog d1=new Dog();
d1.eat();

whereas, when a type of the object is determined at run-time it is known as
late binding.

Animal a=new Dog();

var-args

The var-args allows the method to accept zero or multiple arguments

ENUMS

The Enum in Java is a data type that contains a fixed set of constants.

Enum and Inheritance:

- All enums implicitly extend java.lang.Enum class. As a class can only extend one parent in Java, so an enum cannot extend anything else.
- enum can implement many interfaces.

Methods : values (get array of enums) || ordinal (get enum by index)

Example : 1

```
public enum Item {

        SUGAR,RICE{
                public void info() {
                        System.out.println("This is Rice");
                }
        },SALT;
                public void info(){
                        System.out.println("This is grocery item");
                }
        }
```

Example : 2

```
enum Season{
        WINTER(10),SUMMER(20);
        private int value;
                Season(int value){
                this.value=value;
                }
        }
```

DAY-9

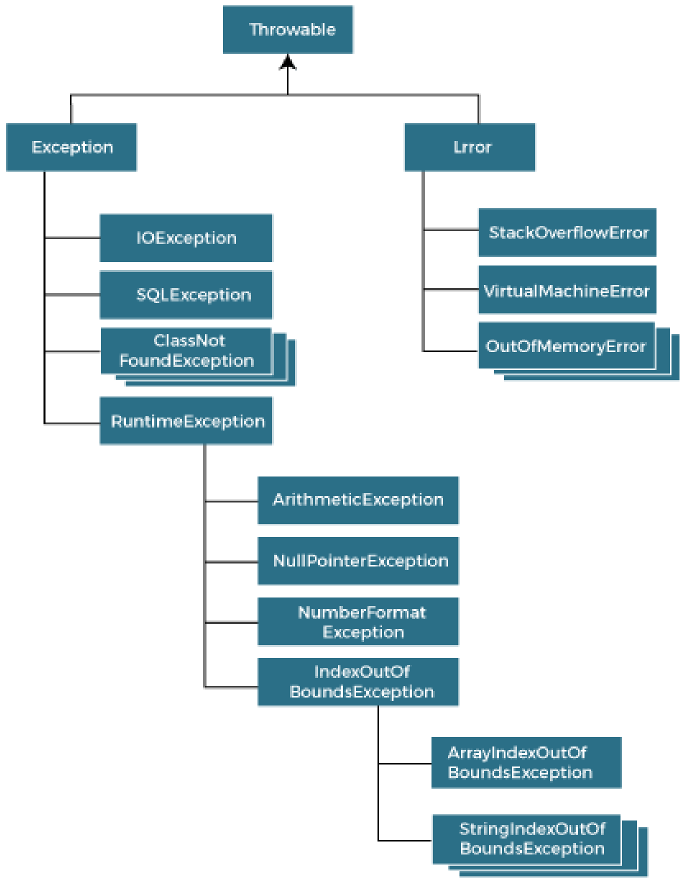
Regular Expression

A regular expression is a special sequence of characters that helps us to match or find other
String

```
Pattern p = Pattern.compile(“ab”);
Matcher m = p.matcher(“abababab”)  => m.start() , m.end();
```

Exception Handling

An unexpected event that is occurring during the execution of program is called Exception
The core advantage of exception handling is to maintain the normal flow of the application.



Checked Exception: The classes that directly inherit the Exception class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.

Unchecked Exception: The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime

Finally : The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. finally block in Java can be used to put "cleanup" code such as closing a file, closing connection, etc.

Throw : The "throw" keyword is used to throw an exception.

Throws : The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method Signature.

Note: For each try block there can be zero or more catch blocks, but only one finally block.

Note: If our User-defined exception class extends from the **java.lang.Exception** class then our custom exception class will become the **checked-exception** class.
whereas if our user-defined exception class extends from the **java.lang.RuntimeException** class then our custom exception class will become the **unchecked-exception.**

DAY-10

Handling Date and Time

1. **java.time.LocalDate:** It represents a year-month-day in the ISO calendar and is useful for representing a date without a time. It can be used to represent a date only information such as birth date or joining date.

2. **java.time.LocalTime:** It deals in time only. It is useful for representing the time of day, such as movie times, or the opening and closing times of the local library.

3. **java.time.LocalDateTime:** It handles both date and time, without a time zone. It is a combination of LocalDate with LocalTime class.

14. **java.time.Duration:** It is used to define the difference between times in time-based values (hour, minute, second, nanoseconds).

5. **java.time.Period:** It is used to define the difference between dates in date-based values (years, months, days)..

6. **java.time.format.DateTimeFormatter:** It comes up with various predefined formatter, or we can define our own. It has a parse or format method for parsing and formatting the date-time values.

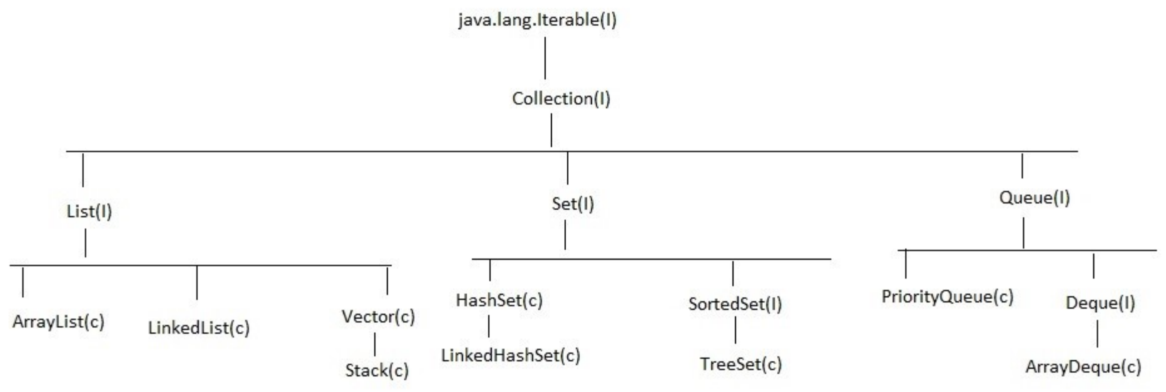
ChronoUnit.YEARS.between

Day-11

Collection framework (I)

A Collection represents a group of objects as a single unit.

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.



Day-12

List interface

List interface is the child interface of the Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects.

List interface is implemented by the classes **ArrayList, LinkedList, Vector, and Stack.**

- List is index-based, it able to arrange all the elements as per indexing.
- List is able to allow duplicate elements.
- List is following insertion order.
- List is able to allow any number of null values.

Auto boxing and Auto unboxing

The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing.

The main advantage of Autoboxing and unboxing is, no need of conversion between primitives and Wrappers manually

so less coding is required.

1. ArrayList:

The ArrayList class implements the List interface. It uses a dynamic array to store the elements.

It dynamically increase and decrease in size.

ArrayList is the best choice if our frequent operation is retrieval.

2. LinkedList class:

This class is another implementation of the List interface, it internally uses the doubly linked list data structure. we can add and remove data from both end.

This class is almost the same as ArrayList class, i.e. it also maintains the insertion order, and allows the duplicate element.

LinkedList class is the best choice if our frequent operation is insertion or deletion of the elements from the middle because no shifting is required.

3. Vector

This class is also one of the implementation classes of List interface.

This class is also same as the ArrayList class with following differences:

Methods of the **ArrayList class is not synchronized,** whereas most of the methods of the **Vector class is synchronized.**

It is a legacy class.

4. Stack

The stack is the subclass of the Vector class. It implements the last-in-first-out data structure.

It is a legacy class.

Set Interface:

This interface extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items.

This Set is implemented by **HashSet, LinkedHashSet, and TreeSet classes.**

HashSet class implements Set Interface.

It does not allow the duplicate elements and Insertion order will not be preserved.

At most one null value we can add in HashSet object.

If our frequent operation is searching, then HashSet class is the best choice, because it internally uses hashing technique to store the objects.

O(1):- is the time complexity of searching any element by using hashing.

LinkedHashSet class:

It is the child class of the HashSet class .

This class is similar to the HashSet class, but it will preserve the insertion order.

TreeSet class:

This class is the implementation of **Set and SortedSet** interface.

In this implementation, objects are sorted and stored in ascending order according to their natural order.

The TreeSet uses a **self-balancing binary search tree**.

Java TreeSet class contains unique elements only like HashSet.

Java TreeSet class **doesn't allow null element.** even a single null is also not allowed otherwise it throws **NullPointerException at runtime**.

Map Interface

HashMap class:

Java HashMap class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the value of the corresponding key.

Elements will be inserted based on hash code of the "key".so it does not follow the sequence.

We can add only one null value as a key, but any number of null can be added as value.

LinkedHashMap class:

It is a child class of LinkedHashMap, and it will preserve the sequence of all the entries

TreeMap class:

TreeMap class implements the **Map and SortedSet interface,** here map will be sorted according to the **natural ordering of its keys**, or by a Comparator provided at map creation time, depending on which constructor is used.

If we don’t use Comparator, then the key object must be Comparable. Otherwise it will raise a ClassCastException.

The TreeMap in Java **does not allow null keys** (like HashMap) and thus

a NullPointerException is thrown. However, multiple null values can be associated with different keys.

Java Hashtable class:

HashMap and Hashtable both are used to store data in key and value form. Both are using hashing technique to store unique keys.

But there are following main differences between HashMap and Hashtable class:

1. **HashMap is non synchronized**. It is **not-thread safe** and can't be shared between many threads without proper synchronization code. whereas **Hashtable is synchronized. It is thread-safe** and can be shared with many threads.

2. **HashMap allows one null key and multiple null values** whereas

Hashtable **doesn't allow any null key or value.**

priority queue

A regular queue has a first in first out structure.

Priority queues help consumers consume the higher priority messages first followed by the lower priority messages.

Generics In Java:

- **Type-safety:** We can hold only a single type of objects in generics. It doesn't allow to store other objects.
- **Type casting is not required:** There is no need to downcast the object.
- **Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime.

WildCard

The question mark (?) is known as the wildcard in generic programming. It represents an unknown type. The wildcard can be used in a variety of situations such as the type of a parameter, field, or local variable; sometimes as a return type.

- **Upper Bounded Wildcards** List<? extends Number.

The purpose of upper bounded wildcards is to decrease the restrictions on a variable. It restricts the unknown type to be a specific type or a subtype of that type.

- **Lower Bounded Wildcards** List<? super Integer.

The purpose of lower bounded wildcards is to restrict the unknown type to be a specific type or a supertype of that type.

DAY-14

Multitasking:

Multitasking is the process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU.
Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

Multithreading

Multithreading is a programming concept in which the application can create a small unit of tasks to execute in parallel.

Multithreading in Java is a process of executing multiple threads simultaneously.

What is process?

A process is an application that is under execution on your computer. It is heavy weight.

- Each process has an address in memory.
- It does not share same memory.
- Context-switching between process takes more time.

What is a thread?

A thread is a lightweight sub-process, the smallest unit of processing.

Thread share same memory area.

Context-switching between thread takes less time as it is happening in same memory.

Note: At least one process is required for each thread.

Mulithreaded application

In Java, every program/application has a default flow of execution, a default thread, it is called the main thread. if we can start another flow of execution(another thread) along with the main thread simultaneously

then it is called a multithreaded application or program.

Runnable Integerface

This Runnable interface belongs to **java.lang package**.

it has only one abstract method: **public void run();**

we can either **implement Runnable interface** (or) **extend Thread class.**

```
class A implements Runnable{
        @Override
        public void run(){
        //define the taks which we want to execute as a thread
        }
}
//or
class A extends Thread
{
        @Override
        public void run(){
                //define the taks which we want to execute as a thread
        }
}
```

Thread scheduler

Thread scheduler in Java is the component of JVM that determines the execution order of multiple threads on a single processor (CPU). It decides the order in which threads should run. This process is called thread scheduling in Java.

Responsibility of start() method of Thread class:

The start() method present in the Thread class is responsible to perform all the mandatory activity which are required for our thread. (like, registering our thread with the Thread-scheduler, performing all the low level task to start a

separate flow of execution. and then calling the run() method).

After starting a thread we are not allowed to restart the same thread once again otherwise we will get a runtime exception called **IllegalThreadStateException.**

```
class ThreadA implements Runnable{
@Override
public void run(){
        for(int i=20;i<40;i++){
                System.out.println("inside run() of ThreadA"+i);
        }
}
public static void main(String[] args){
        ThreadA t1=new ThreadA();
        Thread t=new Thread(t1); //passing Runnable object to the constructor of Thread class
        t.start();
        for(int i=20;i<40;i++) {
                System.out.println("inside main of ThreadA:"+i);
                }
        }
}
```

If we extends Thread class to create a new thread then we loose the chance of Object Orientation biggest advantage of inheritance i.e. we cannot extends another class simultaneously.

To solve the above problem, we use the Runnable interface, so we take a separate class by implementing Runnable interface and overrides the run() method and inside the run() method define the functionality which we want to execute as a separate thread,
then supply the object of that class to the Thread class constructor and create Thread class object and start that thread.

Life-cycle of a Thread

1. New state -> Thread t = new Thread(object);

2. Runnable state -> t.start();

3. Running state -> when selected by thread scheduler it is in running state

4. Blocked state -> if we call sleep(), wait(), join() it is blocked state.

5. Dead state -> when the run method is completed it is DeadState.

Thread name:

Inside Run method

```
String name = Thread.currentThread().getName();
```

Inside main method

t.setName("");

Thread Priorities:

Every thread in java has some priority. it may be default priority generated by the JVM or explicitly provided by programmer.

The valid range of thread priority is 1 to 10. avg – 5

setPriority(int priority);

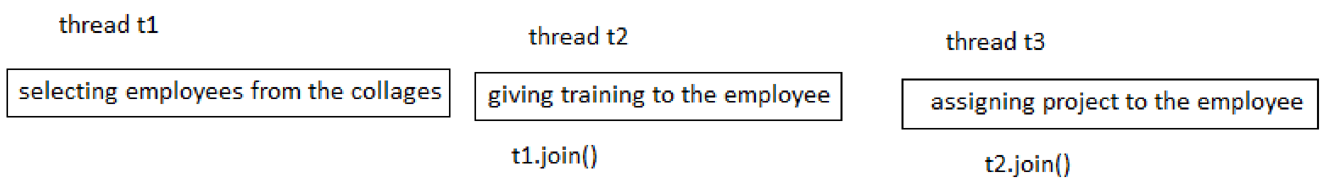
Suspending a thread

It is used to control the execution of the thread, it may be needed in some cases.

Thread.sleep(1000); -> suspend thread for 1000milliseconds - 1 sec

If we want to suspend a running thread conditionally then we should use join() method of the Thread class.
If a thread wants to wait until completion of another thread then we should use join() method.

t.join();


thread t1
selecting employees from the collages
thread t2
giving training to the employee
t1.join()
thread t3
assigning project to the employee
t2.join()

Thread-safety

The concept of avoiding multiple threads acting upon the same functionality simultaneously is known as Thread-safety.

If multiple threads are trying to operate simultaneously on the same functionality then there may be a chance of data inconsistency problem.

Synchronization

1. Internally synchronization concept is implemented by using lock concept.
2. Every object in a java has a unique lock. most of the time the lock is unlocked.
3. It is **object level & class level,** not method level.
4. If we make the method **static** it will get **class level block.**
5. A thread can enter into a synchronized method or block only when If that thread have the lock of that object.
6. The thread won't release the lock until it completes the synchronized methods
7. Until the lock is released (completion of synchronized method.)no other threads can enter any of the synchronized methods or blocks of that object.
8. Acquiring and releasing the lock internally taken care by JVM, programmer are not responsible for this activity.

Note: while one thread is executing any one synchronized method on the given object then the remaining threads are not allowed to execute any other synchronized method simultaneously. but remaining threads are allowed to execute any non-synchronized method simultaneously.

Object Level Lock

```
class A{
synch funA(){}
synch funB(){}
funC(){}
```

}
Here if one thread try to execute funA on A class object then it needs the lock of that object, once it acquires the lock it starts the execution of that method funA, while executing funA by one thread, other threads are not allowed to execute funA and even funB() also, but other threads can executes funC() simultaneously.

if there are two threads try to operate on "two different objects" of Common.java class, then we will get the irregular output even though fun1() is synchronized, because both threads operate on two different objects, and they are holding the locks of two different objects.

Class Level lock:

In java as there is a unique lock for each object of a class, similarly there is a unique lock for each class also.

So there are two types of lock in java:

1. object level lock(it is unique for each object of a class)
2. class level lock(it is unique for each class)

If a thread try to execute a static synchronized method then it required class level lock.

object lock and class level lock both are independent and there is no link between them.

While a thread is executing a static synchronized method, the remaining threads are not allowed to execute any other static synchronized method of that class simultaneously (**even on the multiple object of that class also) but remaining threads are allowed to execute normal static and synchronized non-static methods and normal non-static methods simultaneously.

Synchronized block:

There are following syntax of the synchronized block:
1. synchronized block to get a object level lock of the same class:
Example
```
void fun1() {
        System.out.println("do something...");
        synchronized(this) {--//here if a thread gets the lock of current obj then it is allowed to
        execute
        //this block
        System.out.println("do some more thing1");
        }
}
```
2. synchronized block to get a object level lock of different object
Example:
```
a1=new A();
A a2=new A();
void fun1() {
        System.out.println("do something...");
        synchronized(a1) {//if a thread gets the lock of a1 object of A class
        //(not a2 obj of A class) then only it is allowed
        //to execute the following block of code
}
```
3. To get a class level lock:

Example

```
synchronized(A.class){ //if a thread gets the class level lock of class A
        //then only it is allowed to execute following block
        System.out.println("do some more thing1");
}
```

Race-condition:

the same resource may be accessed by multiple threads at the same time and may change the data. This leads to data inconsistency.

It is avoided with synchorization, we can achive synchonization with the help of **synchronized** keyword.

Thread-synchronization (or) Inter-thread Communication

- It means two synchronized threads communicate with each other.
- Two synchronized thread can communicate each other by using some methods present in Object class, those methods are wait(), notify(), notifyAll().
- To call wait() or notify() method on any object we must have that particular object lock otherwise we will get a runtime exception called IllegalMonitorStateException.

Whenever we need to suspend a synchronized thread unconditionally then we use wait() method.
Whenever we need to resume a suspended(waiting) thread then we use notify() method.
this is known as thread-synchronization or inter-thread communication.

Note:- we can call wait(), notify(),notifyAll() only in the synchronized block or synchronized methods. otherwise we will get a runtime exception.

Deadlock:

- A lock without a key is nothing but deadlock.
- If two threads are waiting for each other forever(for infinite time).
- The synchronized keyword is the only reason for deadlock.

Executor Framework or ThreadPool:

```
Runnable task = () -> System.out.println(“Hello World “ );
Thread thread= new Thread(task);
thread.start();
```

In the above example:
1. A task is an instance of Runnable
2. The task is passed to a new instance of Thread
3. The Thread is launched
4. The thread is created on demand ,by user
5. Once the task is done, thread dies.

6. Thread are expensive resource.

How can we improve the use of Threads ,as a resource ?

Without Thread pool, if we have 10 different independent tasks there then we need to create 10 separate threads.

Thread pool is a pool of already created threads ready to do our tasks.
Thread pool framework is also known as executor framework. this concept is introduced in java5.
Thread pool related API comes in the form of java.util.concurrent package.

```
ExecutorService service=Executors.newFixedThreadPool(3);

for(PrintJob job:jobs){
service.submit(job);
}

service.shutdown();
```

Callable and Future:

In the case of Runnable tasks ,Threads won't return anything after completing the task.
If a thread is required to return some result after execution, then we should use the Callable Interface instead of Runnable.
Callable interface belongs from java.util.concurrent package.

```
public Object call()throws Exception
```

if we submit a Callable object to ExecutorService object, then after completing the task, thread returns an object of the type Future.

Note:- the object of Callable we can't pass to the normal Thread class constructor unlike Runnable object, here we need to use the ExecutorService class help.

Suspending a thread unconditionally:

yield() method:
This yield() method is a static method defined inside the Thread class, it is used to pause the current executing thread for giving the chance to remaining waiting thread of same priority, if there is no any waiting thread or all waiting threads have low priority then same thread will get the chance once again for the execution.

suspend() method:
In order to suspend a thread unconditionally, we make use of non-static method suspend() present in a Thread class.

resume() method:
In order to resume a thread which has been suspended long back we use resume() method, it is also a non-static method defined inside the Thread class.

Note: suspend() and resume() method are deprecated methods. we should not use those methods. we should not use any deprecated methods.

The alternate methods are wait() and notify(). these methods are the non-static method defined inside the Object class.

Difference between yield() and wait() method:
The wait() method will suspend a thread till notify() method is called, whereas yield() method says right now it is not important please give the chance to another thread of same priority.