# Solid Principal:

**main goal is to reduce dependencies so can we can change one area of software without effecting another , these principal are lead to better codes**

**S: single Responsibility principal**

**one class only have a one responsibility.**

**Ex. POJO follow this**

**We do not write programming logic in pojo class**

**Pojo only objectives to have a data members like student class can only contains sname and id but not the address for that we can make another class**

**Also we don’t write programming logic in single class we have seprate class for service dao repo etc.**

**So breaking God Classes into a smaller classes or module is a good practice**

**O : open and closed principal**

**Software components should be open for extension but should be closed for modification**

**Open for extension means : we can extend and include extra functionalities without altering or disturbing our exiting code.**

**Close for modification : means after adding extra functionality we should not modify the existing code**

**Like we make some new changes in code and because of that some old functionality breakes to avoid that we should use this principal**

**Ex. Use of constructor in child class with super**

**L : liskov substitution principal**

**it states that software should not alter the desirable result when we replace a Parent type with one of its base type**

**it requires object of the subclass behave in same way as the object of superclass**

**to avoid misusing of inheritance**

**I : interface segregation principal**

**It states that client should not forced to implement unnecessery methods which they not gonna use.**

**It states that we should split our interface into smaller interfaces and more spacefic once**

**D: dependency inversion principal**

**It allows programmer to remove hardcoded dependencies so that the application become loosely coupled and extendable**

**Ex. We have a employee class which requires object of address class in constructor for initialization so for this we stated this as new address() in constructor but it will create tight coupling between employee and address**

**So to overcome this we can use constructor injection and make is loosely coupled.**

# SDLC :

**It is most widely used approach in software engineering.**

**It follows a number of sequential phases :**

**1.Planning**

**2. Requirements Analysis**

**3. Design**

**4. Development**

**5. Testing**

**6. Deployment**

**7. Maintenance and improvement.**

# Waterfall model:

* **Conception, initiation, analysis, design, construction, testing, implementation and maintenance.**
* **It is set of steps will be done in sequential manner (one after another).**
* **Once a step is completed, you can’t go back to the previous step.**
* **If you make a little change, the whole project will start from zero. So, there's no room for error or change.**

**When to use waterfall:**

* **If the client has complete knowledge of what they want (size, cost & timeline of project), then go for waterfall.**

**Advantages:**

* **so for new resources will it will easy to understand the current situation of the project.**
* **Client knows what the final product will look like.**

# Agile:

* **In this Developers start with simple design and then begin to work on small modules.**
* **The work on these modules is done on a weekly or monthly basis.**
* **After completion of a module, module sent to testing phase,**
* **and if any bug comes, then developer first removes that bug**
* **and then result is deployed in order to take client review,**
* **if client demands any change then first developer has to implement that change.**
* **At the end of each module, project priorities are evaluated, on which module we should start work.**

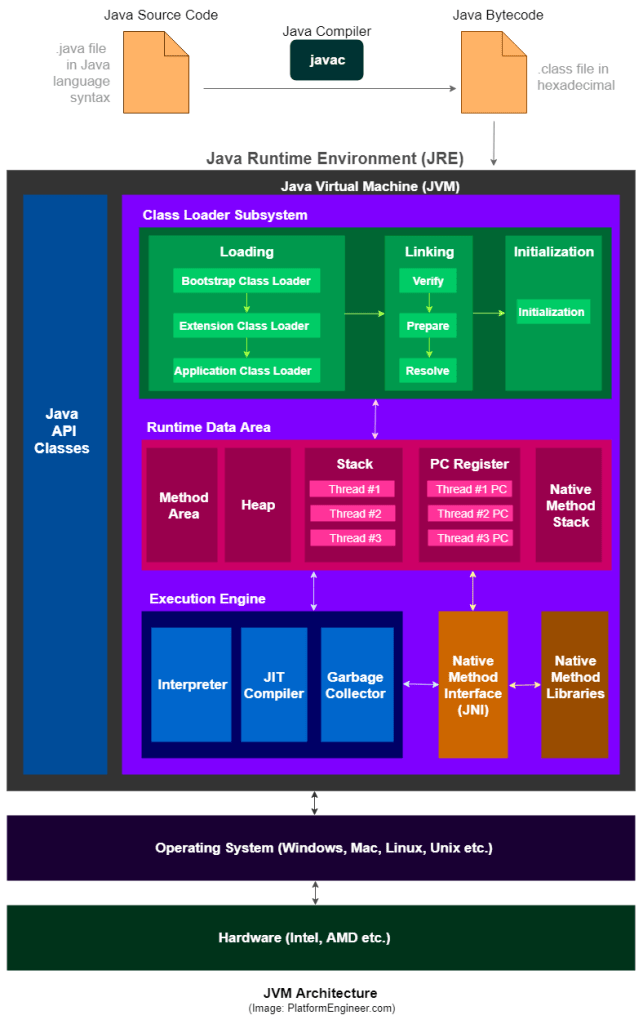
**When to use agile:**

* **When rapid production is more important than the quality of the product.**
* **When there's no clear picture of what the final product looks like.**

**Advantages:**

* **Each module is tested after its completion that trains developers not to make such mistakes in the next module.**
* **Agile allows developers and clients to add change at any time.**
* **After each module, the client reviews the application, so the client knows about the progress of the project after each module.**

# Java Memory Model :

****

|  |  |
| --- | --- |
| Permgen | Metaspace |
| * (Permanent Generation) is a special heap space separated from the main memory. * The JVM keeps track of class metadata in the PermGen. Also, the JVM stores all the static content in this. * Due to limited memory size, PermGen can throw OutOfMemoryError. * It is used in java 7 * Garbage collection of the dead classes and classloaders is triggered once the class metadata usage reaches the MaxMetaspaceSize. | * Metaspace is a new memory space. * It has replaced the older PermGen memory space. * It can now handle memory allocation. * Metaspace grows automatically by default. * It is used in java 8 * The Metadata was is no longer contiained to the Java heap, it has now moved to native memory to an area known as the Metaspace. |

# Class :

**there are no limitations to declare classes in java programe but only one class can be public.**

**For every programme there will be separate .Class file will be generated.**

**Class level modifiers :**

**Public : no restrictions at all**

**Default : only restricted to package**

**Abstract : can not create object**

**Final : can not inhirate that class but can create object**

# Member/variable modifiers:

## Public,default,private,protected

**public : accessible to all the java world**

**default : if we do not declare any modifier then it will be Default access modifier only acessable in same package**

**Private : only accessible in same class.**

**Protected : default + child class**

## **Volatile :**

**By declaring the variable volatile it will make sure :**

1. **all writes to that specific variable will be written back to main memory immediately.**
2. **Also, all reads of that variable will be read directly from main memory.**

**Ex.**

**So there is situation in which two or more threads have access to a one class**

**which contains integer A;**

**public class Test{**

**public int A = 0;**

**}**

**So In that only Thread 1 increments the value of A**

**but there are another thread like Thread 1 and Thread 2 may read the variable A from time to time.**

**So If the counter variable is not declared volatile there is no guarantee about :**

**when the value of variable A is written from the CPU’s cache back to main memory.**

**This means, that the variable A value in the CPU cache may not be the same as main memory.**

## **atomic:**

**so By atomic, mean each action takes place in one step and cannot be interrupted.**

**Because of this , when we have multithreaded applications :**

**the read and write operations are thread-safe and need not be synchronized.**

**Ex. simply our getters and setters methods are thread safe**

# Object Class methods :

**toString():**The toString() provides a String representation of an object and is used to convert an object to String.

**hashCode():** For every object, JVM generates a unique number which is hashcode.

It converts the internal address of the object to an integer by using an algorithm.

* **Use of hashCode() method:**It returns a hash value that is used to search objects in a ––collection.

JVM uses the hashcode method while saving objects into hashing-related data structures like HashSet, HashMap, Hashtable, etc.

The main advantage of saving objects based on hash code is that searching becomes easy.

* **Override of hashCode() method** : needs to be done such that for every object we generate a unique number. For example, for a Student class, we can return the roll no. of a student from the hashCode() method as it is unique.

**equals(Object obj):** It compares the given object to “this” object (the object on which the method is called).

is recommended to override the **equals(Object obj)** method to get our own equality condition on Objects.

Ex.

@Override

**public** **boolean** equals(Object obj) {

**if** (obj == **null** || getClass() != obj.getClass()) { //obj of current class or not

**return** **false**;

}

**if** (obj == **this**) { //memory location is same or not

**return** **true**;

}

Stu10 stu = (Stu10)obj;

**return**(stu.rollNo == **this**.rollNo); // if this is also returning true

}

**HashCode Equals Contract :**

**If two objects are equal according to .equals(obj o) method then hashcode of both the objects must be same.**

**If two objects has same hashcode it is not necessary means that both are equal it is condition of collision.**

**Whenever we invoked same object more then once during an execution,the hashcode method must return the same result or integer**

**getClass():** It returns the class object of “this”/currently executing object and is used to get the actual runtime class of the object.

Ex.

public class Test {

public static void main(String[] args){

Object obj = new String("GeeksForGeeks");

Class c = obj.getClass();

System.out.println("Class of Object obj is : "

+ c.getName());

}

}

**finalize() method:**This method is called just before an object is garbage collected.

It is called the [Garbage Collector](https://www.geeksforgeeks.org/garbage-collection-java/) on an object when the garbage collector determines that there are no more references to the object.

**clone():** It returns a new object that is exactly the same as this object.

* **Shallow Cloning:**  
  **in shallow cloneing the object copies only ‘main’ object, but doesn’t copy the inner objects.**

**When a custom object like ( Employee) if it only has just primitive, String type variables then we use Shallow Cloning.**

**Employee e = new Employee(2, "sid");**

**Employee e2=e.clone();**

* **Deep Cloning**:  
  **Unlike the shallow copy, a deep copy is a fully independent copy of an object."**

**Like if an Employee object holds an Address object :**

**Then we have to write the code to clone the 'Address' object also in the overridden clone() method. Otherwise the Address object won't clone and it causes a bug when you change value of Address in cloned Employee object, which reflects the original one too.**

**Employee e = new Employee(2, "sid", new Address(7, "colony", "city");**

The remaining three methods **wait()**, **notify()** **notifyAll()** are also in object class because of synchronization concept is depend on lock and locks are depend on objects

# Construtor :

**It is specially design to initialization of object**

**Student s1 = new Student(“sid”,22);**

**name of the class and name of constructor must be same**

**return type concept not applicable for constructor**

**constructor modifier :**

**public : can create object anywhere**

**default : only create obj in same package**

**protected : default + also create obj in child classes**

**private : only one obj is allowed to create**

* **every no arg constructor is not default constructor.**
* **Access modifier of default constructor is same as class modifier**
* **Default constructor only contains one line**
* **super() - it is no-args call to super class constructor**
* **first line inside every constructor is always super()/this().**
* **We can only call constructor with the help of another constructor not with method**
* **It is possible to overload constructor in same class**
* **Inheritance is not applicable on Constructor**
* **If there is any chance of recursive call compiler will check and not execute the programme**
* **If parent class contains args constructor ,while writing child classes we have to take special care**
* **When we writing any args constructor there is high need to write one no args constructor**

# Final vs finally vs finalize

|  |  |  |
| --- | --- | --- |
| Final | Finally | finalize |
| * **it is a modifier applicable for classes** * **if a method declare final we can not override that method** * **if a variable declared as final we it will become constant** | * **It is a block always associaited with try and catch to maintain cleanup code** | * **Is a method always invoked by garbage collector just before destroying an object to perform cleanup** |

# String

**A String is a sequence of characters.**

**In the Java, strings are objects.**

## When we say, string are immutable :

We are actually taking about string object in memory

But string variables can point to any string we want

**the String is a Java Class defined in**[**java.lang package**](http://www.javaguides.net/p/java-lang-package-tutorial.html)**.**

**It’s not a**[**primitive data type**](http://www.javaguides.net/2018/10/primitive-data-types-in-java.html)

**There are two ways to create a String object:**

* 1. **By string literal (String str = “abc”) :**

**In this case, string literal (“abc”) will be put it in SCP and variable str will point to it.**

**2.By new keyword (String str = new String(“abc”))**

in such a case, JVM will create a new string object in heap memory and the literal “xyz” will be placed in the string constant pool(SCP).

The variable str will refer to the object in the heap memory

**What do you mean by mutable and immutable objects?**

**Immutable objects are like constants. You can’t modify them once they are created. They are final in nature.**

**mutable objects , you can perform modifications to them.**

What is String Constant Pool ?

String Constant Pool is the memory space in heap memory

specially allocated to store the string objects created using string literals.

**Does String is thread-safe in Java?**

**Yes, Strings are immutable in Java so once we create a String object then we can't change its content. Hence it’s thread-safe and can be safely used in a multi-threaded environment.**

Difference between == and equals() method in Java

**The main difference between the *equals()* method and *==* operator is that one is a method and the other is an operator.**

**We can use *==* operators for reference comparison (address comparison)**

**and *equals()* method for content comparison.**

**In simple words, == checks if both objects point to the same memory location**

**whereas *equals()* evaluates to the comparison of values in the objects.**

What does the String intern() method do?

The method *intern()* creates an exact copy of a String object in the heap memory and stores it in the String constant pool.

12. Why String is a popular HashMap key in Java?

Since String is immutable, its hashcode is cached at the time of creation and it doesn’t need to be calculated again. and it’s processing is fast than other HashMap key objects. This is why String is mostly used Object as HashMap keys

# String vs string buffer vs string builder

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **String** | | **String buffer** | **String builder** |
| **Storage** | **Heap ,scp** | | **heap** | **heap** |
| **Object** | **immutable** | | **mutable** | **mutable** |
| **Memory** | **Takes lot memory** | | **Less memory** | **Less than string buffer** |
| **Thread safe** | **thread safe** | | **All methods are synchronized thus it is thread safe** | **Contains non synchronized methods so not thread safe** |
| **performance** | **slow** | | **Fast compare to string** | **Fast compare to string buffer** |
| **Use** | **When data is not changing frequently** | **When data is changing more frequently** | | **-** |

# Java programe to check occuernace of word

public class CheckOccuerance {

public static void main(String[] args) {

String string = "java is fun i love java it is easy to learn";

String[] arrstr = string.split(" ");

Map<String, Integer> hMap = new HashMap<>();

for (String word : arrstr) {

if (hMap.get(word) == null) {

hMap.put(word, 1);

}else {

hMap.put(word, hMap.get(word)+1);

}

}

System.*out*.println(hMap);

Iterator<String> iterator = hMap.keySet().iterator();

while (iterator.hasNext()) {

String w = (String) iterator.next();

if (hMap.get(w)>1) {

System.*out*.println(w+" : "+hMap.get(w));

}

}

}

}

# Write a Java Program to Check if Input String is Palindrome

public class CheckPAlindrome {

public static void main(String[] args) {

String string = "mom and dad coming at noon";

String string3 = "java k";

String[] split = string.split(" ");

for (String string2 : split) {

if (*checkWord*(string2)) {

System.*out*.println(string2);

}

}

static boolean checkWord(String str) {

String wordOne = "";

for(int i = str.length()-1;i>=0;i--) {

wordOne = wordOne + str.charAt(i);

}

if (str.equals(wordOne)) {

return true;

}else {

return false;

}

}

}

}

# Write a Java Program to Remove Duplicate Words from String

public class RemoveDuplicateWords {

public static void main(String[] args) {

String s1 = "java is java i love java";

String[] split = s1.split(" ");

Set<String>set = new LinkedHashSet<>();

for (String string : split) {

set.add(string);

}

StringBuilder builder = new StringBuilder();

for (String string : set) {

builder.append(string).append(" ");

}

System.*out*.println(builder);

}

}

# Write a Java Program to Reverse a String(5 ways)

**public class ReverseEachWord {**

**public static void main(String[] args) {**

**String s1 = "my name is siddhesh";**

**String s2 ="";**

**String[] split = s1.split(" ");**

**for (String string : split) {**

**String reverse = *reverse*(string);**

**s2 = s2.concat(reverse).concat(" ");**

**}**

**System.*out*.println(s2);**

**}**

**private static String reverse(String word) {**

**String s2 = "";**

**for(int i = word.length()-1;i>=0;i--) {**

**s2 = s2 + word.charAt(i);**

**}**

**return s2;**

**}**

**}**

# How to Check If the String Contains Only Letters or Digits?

**public class IsLetterOrDigit {**

**public static void main(String[] args) {**

**String s1 = 8211+"";**

**System.*out*.println(s1);**

**boolean flag = true;**

**for(int i = 0;i<s1.length();i++) {**

**if (!Character.*isDigit*(s1.charAt(i))) {**

**flag = false;**

**}**

**}**

**if (flag == false) {**

**System.*out*.println("not contains only digits");**

**}else {**

**System.*out*.println("contains only digit");**

**}**

**}**

**}**

# Static

**static is not access modifier.**

**It can be only used with class level variable not with local variable.**

**It can also used with :**

* **Method**
* **Block**
* **Inner class (nested class)**

**It belongs to class not objects**

**We have to call Static variables using class Name**

**Static variables are used for memory management**

**Static variable takes space in class/method area**

**Static Method :**

**Static Method belongs to class not an obj.**

**Can be called by directly class name.**

**If we calling static method within a class there is no need to use classname.**

**Static method can only access static data.**

**Static method can only call static methods**

**We can not use this or super keywords with static method**

**Static Block :**

**There is no way to call static block when class is loaded in memory static block automatically gets called**

**We can create multipal static block in a single class**

**Static block is used to initialize static member.**

# Array

**An Array is a data structure that defines an index-based collection of a same type of elements.**

**Whose index is starts from zero**

**Advantages :**

**The main use of Array is used to store multiple values in a single variable, instead of declaring separate variables for each value.**

**We can access any element randomly by using indexes provided by arrays.**

**We can sort multiple elements of Array at the same time.**

**Disadvantages :**

**Size Limit: We can store the only fixed size of elements in the array. It doesn't grow its size at runtime.**

**Arrays are Strongly-Typed this means that all elements in the array have the same data type. We can not store different types of data in an Array.**

**Single Dimensional Array**

**A single dimensional array of Java is a normal array where the array contains sequential elements (of the same type).**

**Multidimensional Array**

**A multi-dimensional array in Java is an array of arrays.**

Can You Pass the Negative Number in Array Size?

**No, you can not pass the negative number as Array size. If you pass a negative number in Array size then you will not get the compiler error. Instead, you will get the NegativeArraySizeException at run time.**

When ArrayIndexOutOfBoundsException occur?

**ArrayOutOfBoundsException is thrown when an attempt is made to access the Array with an illegal index. For example, an illegal index means if the index is either negative or greater than or equal to the size of the Array.**

Where Does Array Store in JVM Memory?

**As we know that Array is an object in java. So, Array is stored in heap memory in JVM**

What is ArrayStoreException? When this exception is thrown?

*ArrayStoreException* is a runtime exception. The array must contain the same data type elements.  
  
This exception is thrown to indicate that an illeagel attempt has been made to store the wrong type of object into an array of objects. In other words, if you want to store the integer Object in an Array of String you will get *ArrayStoreException*.

What is an Anonymous Array in Java?

An array without any name (or reference) is called an Anonymous Array. They are useful for the scenarios where we need one-time usage of Array.

Ex. (new int[]{12,20,30})

# Public Static Void Main(String[] args) :

* Overriding of main method is possible but compiler will always call method with string argument.
* Overloading is not applicable for main method
* As both method are static it is called as method hiding
* If we not written main method properly jvm will check this at run time
* We can not change main method syntax it will cause error
* There are some changes acceptable :
* can change order of modifiers
* we can decleare (String[] args in any acceptable form)
* instead of args we can take any vaild java identifier
* we can replace string[] with (string… args)
* we can decleare main method with following modifiers also:

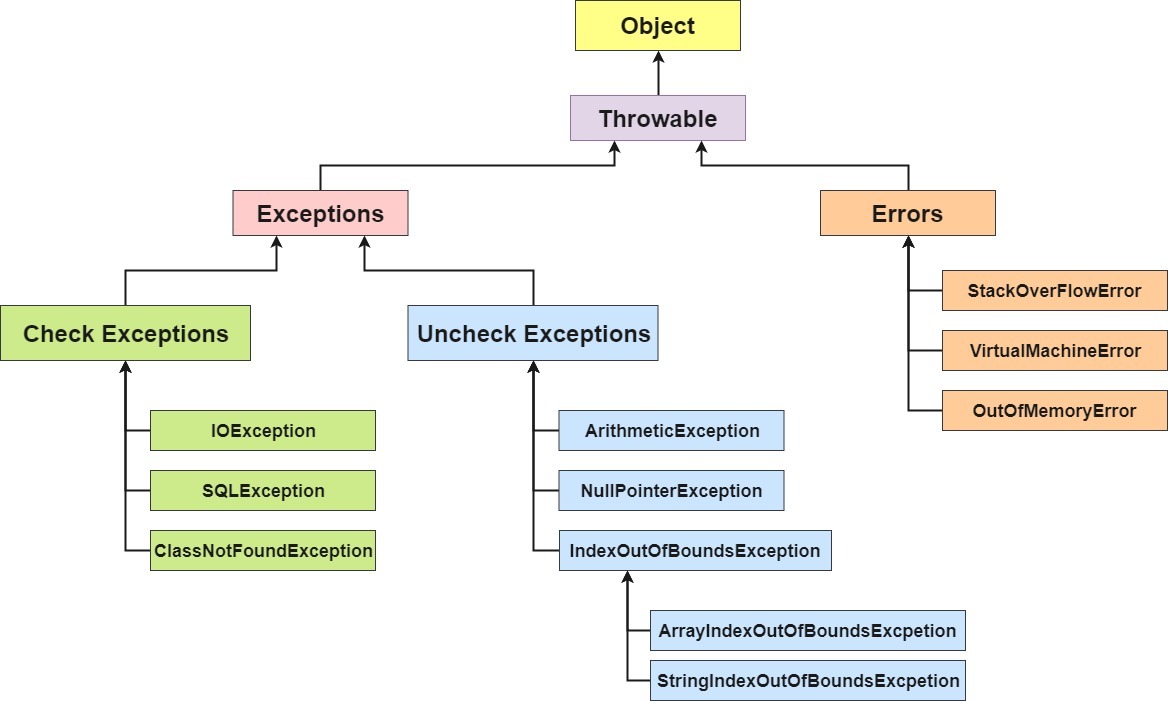
final , synchronized

1. public : to call by jvm from any location
2. static : without any object jvm has to call main method
3. void : main method won’t return anything to jvm
4. main : this name configured inside jvm
5. (String [] args) : these are command line args

# System.out.println() :

1. System : is a class
2. Out : is static variable present in system class of type print stream
3. Println : is a method present inside printStream class

# Exception Handling



**Exception :**

Most of exception are caused by our programme & it is recoverable

Error :

Error are not caused by our programme, most of time it is due to lac of system compatibility & it is not recoverable

## Checked Exception :

The exception which are checked by compiler for the smooth execution of programme at runtime these are called checked exception.

In case of checked exception compiler will check whether we are handling exception, if the programme not handling exception then we will get compile time error

## Unchecked Exception :

The exception which are not checked by compiler are called unchecked exception .

In case of checked exception compiler wont check whether programmer handling exception or not

Whether exception checked or unchecked compulsory it will be occur only at runtime there is no chance of occurring any exception at compile time.

Exception is said to be fully checked if and only if its child exception are also checked exception

If only parent class is checked exception but its some of child are not checked then it said to be partially checked exception

Exception class

Throwable class

The method in which exception rise that method is only responsible for creating exception object and handover to JVM

## **Throw vs throws**

|  |  |
| --- | --- |
| Throw | Throws |
| * Sometimes we can create exception obj explicitly and handover that to JVM manually for this we use throw keyword * We are not allowed to write anything after throw statement otherwise it will cause compile time error * We can only throw exceptions and errors every java obj is not throwable. | * To delegate work or responsibility of exception handling to caller for that purpose we use throws keyword * It is only required for checked exception * The throw keyword is only there to convince compiler it will not prevent abnormal termination of programme * Therefore it is recommended to use try&catch. |

## Customize exception/user define exception :

To create checked Exception :

In this case we have to extend our class with Exception

This is checked exception so we have to handle this with try-catch or throws that custom exception

Ex.

class UnderAgeException extends Exception{

public UnderAgeException() {

super("under age 1");

}

}

**public class TestClass{**

**public static void main(String[] args){**

**int age = 17;**

**if (age < 18) {**

**try {**

**throw new UnderAgeException();**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**}**

**}**

}

To create Unchecked checked Exception :

In this case we have to extend our class with RunTimeException

This is Unchecked exception so we don’t have need to handle this with try-catch or throws that custom exception

Ex.

**class UnderAgeExceptionNew extends RuntimeException{**

**public UnderAgeExceptionNew() {**

**super();**

**}**

**public UnderAgeExceptionNew(String message) {**

**super(message);**

**}**

**}**

**public class TestClass{**

**public static void main(String[] args){**

**int age = 17;**

**if(age<18) {**

**throw new UnderAgeExceptionNew("yoy are under 18");**

**}**

**}**

}

## **Array exceptions :**

**ArrayIndexOutOfBounds Exception :**It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

**NoSuchMethodException : i**t is thrown when accessing a method which is not found.

**NullPointerException :**This exception is raised when referring to the members of a null object. Null represents nothing

**NumberFormatException :**This exception is raised when a method could not convert a string into a numeric format.

|  |  |
| --- | --- |
| classNotFound exception | NoClassDefFound error |
| For hardcoded class name at runtime if the corrsponding .class file not available then we get this exception | For dynamically provide class name at runtime if .class file not available then the following error will cause. |

## Exception propagation :

Public void m1(){

m2();

}

Public void m2(){

Exception!!!!

}

M1() is calling m2() inside, and if there any exception raised in m2() and it will be not handled then that exception in **propgated** to caller method (m1()) .

## Rethrowing Exception:

Converting one exception type to another exception

Try{

Sysout(10/0)

}catch(ArthematicException e){

Throw new nullPointerException();

}

# Association :

It is HAS-A relationship between two objects.

Aggregation :

it is a loose type of association between two objects where both can live independently .

Ex. car has a driver

in this case car can exist without driver and also driver can exist without car.

Composition : in this one object can not exist without another object

It is tight association between two objects

Ex. car has engine

Where IS-A relationship is completely inheritance. This means, that the child class is a type of parent class.

# Serialization :

The process of saving state of object into a file is called serialization

It is also a process of converting object from normal java supported form into a file/network supported form .

It can be done using file output stream and in front of that object output Stream’s .writeObject(obj o ) method is there.

We can serialize only serializable objects, the object is called serializable when class of that object implements serializable interface .

serializable interface is marker interface it does not contain any method

in this we are saving our data permanently for future

## Transiant:

it is a modifier only applicable for variable

At the time of serialization we don’t want to save the value of particular variable, then we go for Transient keyword.

Also At the time of serialization jvm ignores original value of transient variable and save default value to the file

Basically transient means not to serialize.

static variables are not a part of serialization

if we decleare static variable Transient there is no use.

Final variable will be part of serilization , directly by there value

## De-serializaton :

It is the process of reading state of a object from a file .

## Externalizable :

Basically, the difference between [Serializable](http://java.sun.com/javase/6/docs/api/java/io/Serializable.html) and [Externalizable](http://java.sun.com/javase/6/docs/api/java/io/Externalizable.html) is :

the classes which implement Serializable, the serialization of the object is taken care of automatically,

while classes that implement Externalizable is responsible for serializing itself, without the help of default serialization procedures.

# Marker Interface :

It is a interface containing no methods

Ex.

In case of cloneable interface at the time of implementation class object will get some extra services to create clone of object.

Class A implements cloneable{

String str = “hello”;

P s v m(…….){

A a = new A();

A a2 = (A)a.clone();

}

}

# Enum :

If we want to represents group of constants then we should go for enum

Ex .

Enum month{

Jan,feb,march……….,dec ; //semicolon is optional

}

Main objective is to define our own datatype

Internally enum implemented using class concept

Every enum is always public static final

we can declare enum either inside the class or outside the class but can not declare inside method

if declare outside class applicable modifiers are : public default strictfp

if declare inside class applicable modifiers are : public default strictfp private protected static

inheritance concept not applicable for enum but it can implement many interface

Values() : to return all values present inside enum .

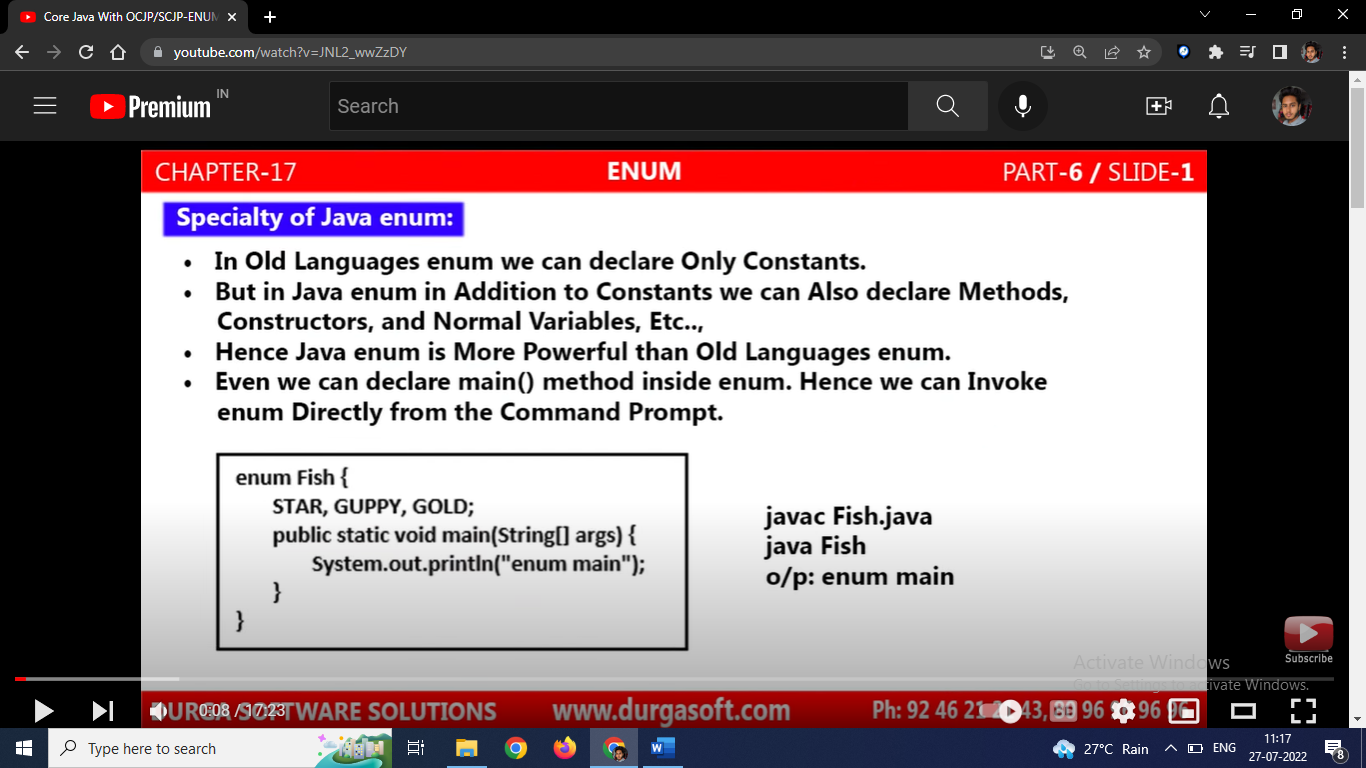
Ex . Month[] m = Month.values();

Ordinal() : to represent order of constant inside enum.

For(Month m1:m){  
 sysout(m1+” : ”+m1.oridinal());

}

We can also declare methods, constructor, normal varibales, main method inside enum.



List of constants always should be first in any enum

Empty enum [ enum Month{}; ] also vaild syntax.

Constructor of enum saperately executed for every element

# OOPS

Object-Oriented Programming System is the programming technique to write programs based on the real world objects. And in those programs states and behaviors of an object are represented as the member variables and methods.

## Abstraction , encapsulation , polymorphism,inheritance

# Interface vs abstractclass

|  |  |
| --- | --- |
| Interface | Abstract class |
| * it is an 100% abstract class which only contains abstract methods * if we don’t know anything about implementation , just we have a requirement specification then we should go for interface * every method present in interface is by default public and abstract * every variable present in interface are always public static finale * no constructor allowed * we cant declare interface method with private, protected, final ,static synchronized , native | * Abstract class is class which can contain both abstract and concrete methods * if we are taking about implementation but not completely (like partial implementaion) then we should go for abstract class * every method need not to be public abstract * no such restriction * we can take constructor * no such resctriction |

Concreate class :

if we talking about implementation completely & ready to provide service then we should go for concreate class.

# Abstraction vs encapsulation

|  |  |
| --- | --- |
| Abstraction | Encapsulation |
| The process of showing necessary information and hiding unnecessary data or implementation is called an "Abstraction" | It is the process of binding data and coding part into a single unit |

Both are associated with data hiding.

Encapsulation :

it is a process of grouping or combining data members and corresponding methods into an single unit

encapsulation = data hiding + abstraction

polymorphism :

it means one name but multipal forms

two types of polymorphism :

compile time (overloading , methodhiding)

runtime(overriding)

# overloading vs overriding

|  |  |
| --- | --- |
| Overloading | Overriding |
| same method name  different argument type  called as compile time polymorphism  in case of overloading if the exact matching type is not available then compiler wont rise compile time error immediately,  first compiler will promote args type to next level  while overloading child will get highest priority  while overloading exact match will get high priority | method name should be same  method signature should be matched  method return type should be same until(1.4v) after that co-varient data types are all allowed  it is not applicable for private methods  not override final method  we can override non-final method to final but after that further overriding not possible  abstract to non-abstract possible  non-abstract to abstract also possible, further extending class have to give implementation of that abstract class  synchronized ⬄ non-synchronized possible  we can not reduce scope of method,we can extend |

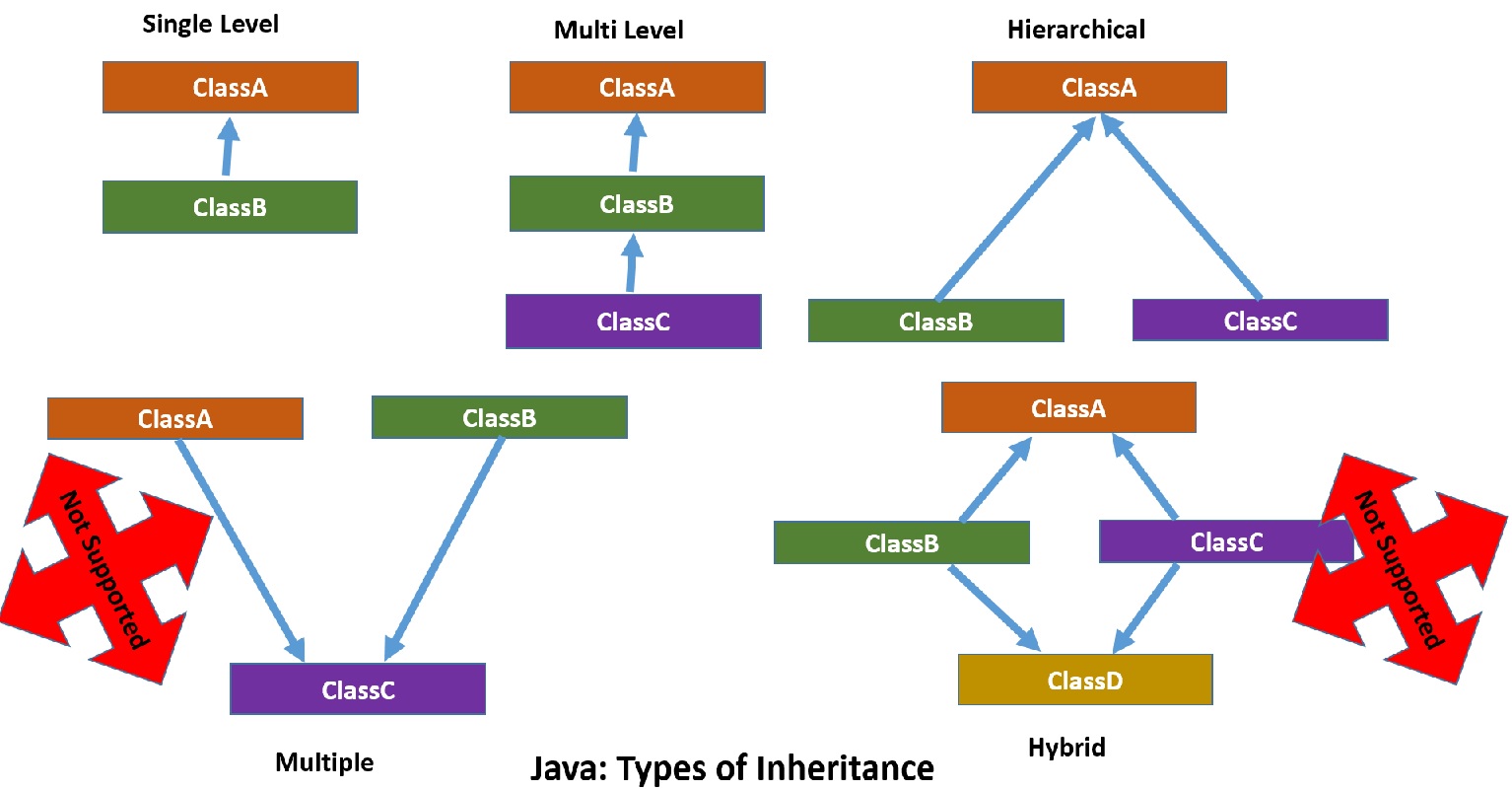
Overriding rules about exception :

If the child method throws any exception compulsory the parent should throw the same checked exception or its parent exception

If parent class throw any checked exception there is not any compulsion for child class method to throw exception

Static ⬄ non-static is not possible it is called method hiding

# Inheritance :



multiple inheritance and hybrid inheritance not supported by java as it will create ambiguity.

If class does not extend any other class then only it it the direct child class of object class

Cycelic inheritance is also not supported by java

# Object :

What is an Object?

**Object is an instance of a class. Also The Object is the real-time entity having some state and behavior.**

## Object typecasting :

**1)the type of object and the type of object we trying to convert must have some relation**

**ex. Parent -child , child-parent, child-child, parent – parent**

**2)typecasted type and the type of object we are trying to save is must be same or its parent**

**ex. StringBuffer sb = (StringBuffer)O;**

**these two rule compiler check at compile time.**

**In typecasting new object is not created only new variable is assign to existing object**

**What is Class?**

**A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.**

# Multi-Threading

**Thread : so basically Thread is a flow of execution to perform a particular task.**

## **Two ways to define thread :**

**Extending thread class (not preferred no further chance of extending that class)**

**Implementing runnable interface (preferable )\***

**Extending Thread class :**

class MyThread extends Thread{

@Override

public void run() {

for (int i = 0; i < 10; i++) {

System.*out*.println(i);

}

}

}

public class TestClass{

public static void main(String[] args) {

MyThread thread = new MyThread();

thread.start();

for (int i = 101; i < 110; i++) {

System.*out*.println(i);

}

}

}

**Implementing Runnable Interface:**

class MyRunnable implements Runnable{

@Override

public void run() {

for (int i = 0; i < 10; i++) {

System.*out*.println("child thread - "+i);

}

}

}

public class TestClass{

public static void main(String[] args) {

MyRunnable runnable = new MyRunnable();

Thread thread = new Thread(runnable);

thread.start();

for (int i = 0; i < 10; i++) {

System.*out*.println("main thread - "+i);

}

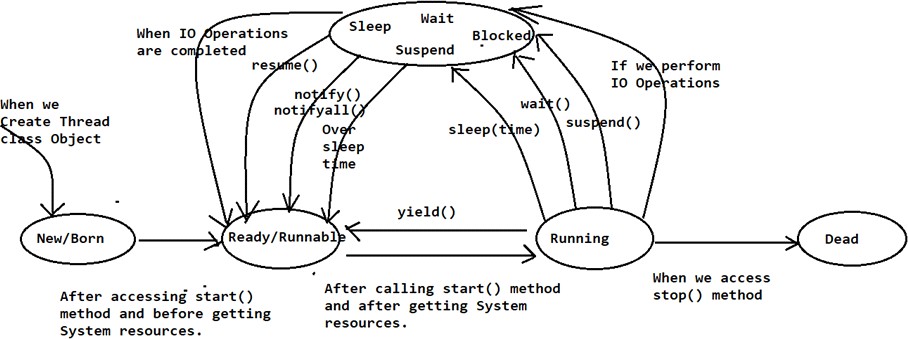
}

}

## Yield() – join()-sleep :

|  |  |  |  |
| --- | --- | --- | --- |
|  | Yield() | Join() | Sleep() |
|  | If thread wants to pause its execution to give the chance for remeaning thread with same priority  Then we go for yield() | If thread wants to wait until some other thread compleating its execution then we go join() | If thread don’t want to perform any operation for particular time then we go for sleep |
| Overloaded | no | yes | yes |
| Final | no | yes | no |
| Throw interrupted | no | yes | yes |
| Native | yes | no | yes |
| Static | yes | no | yes |

## **Thread life-cycle :**

****

## **Thread Class Library:**

**Constructors:**

**public Thread()**

EX: Thread t = new Thread(); System.out.println(t);

**Output: Thread[Thread-0, 5, main]**

**public Thread(String name)**

This constructor can be used to create Thread class object with the specified name.

EX: Thread t = new Thread("Core Java"); System.out.println(t);

**Output: Thread[Core Java,5,main]**

**public Thread(Runnable r)**

This constructor can be used to create Thread class object with the specified Runnable reference.

EX: Runnable r = new Thread(); Thread t = new Thread(r); System.out.println(t);

**Output: Thread[Thread-1,5,main]**

**Methods:**

**public void setName(String name)**

It can be used to set a particular name to the Thread explicitly.

**public String getName()**

It can be used to get thread name explicitly.

EX.

**Thread t=new Thread();**

**System.out.println(t.getName());**

**t.setName("Core Java");**

System.out.println(t.getName());

**public void setPriority(int priority)**

It can be used to set a particular priority value to the Thread, 1-10 any number we can set.

public static final int MIN\_PRIORITY=1;

public static final int NORM\_PRIORITY=5; public static final int MAX\_PRIORITY=10;

ex.

**t.setPriority(7);**

**public int getPriority()**

It can be used to get priority value of the Thread.

**public static int activeCount()**

It will return the no of threads which are in active.

**Public boolean isAlive()**

This method can be used to check whether a thread is in live or not.

**public static thread currentThread()**

It can be used to get Thread object reference which is in active at present.

Ex.

Thread.currentThread().getName();

**public static void sleep(long time) throws InterruptedException**

**This method can be used to keep a running thread into sleeping state up to the specified sleep time.**

**Public Static Native void yield()throws InterruptedException**

This method will pause current executing thread to give the chance for the same priority waiting thread.

Ex.

class MyRunnable implements Runnable{

@Override

public void run() {

for (int i = 0; i < 10; i++) {

System.*out*.println("child thread - "+i);

Thread.yield();

}

}

}

public class TestClass{

public static void main(String[] args) {

MyRunnable runnable = new MyRunnable();

Thread thread = new Thread(runnable);

thread.start();

for (int i = 0; i < 10; i++) {

System.*out*.println("main thread - "+i);

}

}

}

**public void join(long miliseconds)throws InterruptedException**

**This method will pause a thread**

**to complete a thread on which we accessed join() method**

**if the thread calls join method on same thread itself then programe get stucked then this situation calls deadlock.**

## **Daemon Threads**

These threads are running internally to provide services to some other thread and it will be terminated along with the threads which are taking services.

Main thread is not demon thread

We can not make main thread to a demon thread because it is already stored in jvm

**public void setDaemon(boolean b)**

**public boolean isDaemon()**

EX: In Java, Garbage Collector is a thread running internally inside JVM and it will provide Garbage Collection services to JVM and it will be terminated along with JVM automatically.

# Synchronization:

**if we execute more than one thread on a single data item then there may be a chance to get data inconsistency**

**so to avoid that we have to use synchronization**

**so it will only allow one thread at a time.**

**is going on the basis of Locking mechanism.**

**Ex.**

**class SyncDemo{**

**synchronized void m1(){**

**for (int i = 0; i < 5; i++) {**

**System.*out*.println(Thread.*currentThread*().getName()+" "+i);**

**}**

**}**

**}**

**}**

**class Thread1 implements Runnable {**

**SyncDemo demo;**

**public Thread1(SyncDemo demo) {**

**super();**

**this.demo = demo;**

**}**

**@Override**

**public void run() {**

**demo.m1();**

**}**

**}**

**class Thread2 implements Runnable {**

**SyncDemo demo;**

**public Thread2(SyncDemo demo) {**

**super();**

**this.demo = demo;**

**}**

**@Override**

**public void run() {**

**demo.m1();**

**}**

**}**

**class Thread3 implements Runnable {**

**SyncDemo demo;**

**public Thread3(SyncDemo demo) {**

**super();**

**this.demo = demo;**

**}**

**@Override**

**public void run() {**

**demo.m1();**

**}**

**}**

**public class TestClass{**

**public static void main(String[] args) {**

**SyncDemo demo = new SyncDemo();**

**Thread1 thread1 = new Thread1(demo);**

**Thread2 thread2 = new Thread2(demo);**

**Thread3 thread3 = new Thread3(demo);**

**Thread t1 = new Thread(thread1);**

**Thread t2 = new Thread(thread2);**

**Thread t3 = new Thread(thread3);**

**t1.start();**

**t2.start();**

**t3.start();**

**for (int i = 0; i < 5; i++) {**

**System.*out*.println(Thread.*currentThread*().getName()+" "+i);**

**}**

**}**

**}**

## **Synchronized Block:**

**If we need to synchronize only few lines of code then it not recommended to delceare entire method synchronize**

**For those few lines of code we can use synchronize block**

**Ex.**

synchronized(Object o/this/current object reference)

{

-----

-----

}

Java provide us two types of lock

Object lock

Class lock (class\_name.class)

## **Inter Thread Communication:**

The process of providing communication between Threads .

These methods are present in object class.

wait() method can be used to keep a thread in waiting state.

notify() method can be used to give a notification to a thread which is available in waiting state.

notifyAll() method can be used to give a notification to all the threads which are available i waiting state.

These methods are in object class because , these methods work on the locks and locks are associated with objects and not thread ,

|  |  |  |
| --- | --- | --- |
| Producer Thread  Class ProducerThread{  Public void produce(){  Synchronized(Queue){  Produces items in queue  q.notify();  }  }  } | Queue of items | Consumer Thread  Class ConsumerThread{  Public void consume(){  Synchronized(Queue){  If(q.isEmpty){  q.wait();  }else  consumes items in queue  }  }  } |

## **Dead Lock:**

**It is situation where two threads waiting for each other forever.**

**There is no specific solution to that**

**But we can prevent this situation by using synchronization**

**Ex.**

**class A{**

**public synchronized void d1(B b) {**

**System.*out*.println("thread 1 started d1 mehod");**

**try {**

**Thread.*sleep*(1000);**

**} catch (Exception e) {**

**// TODO: handle exception**

**}**

**System.*out*.println("thraed 1 trying to call B's last method");**

**b.last();**

**}**

**public synchronized void last() {**

**System.*out*.println("inside A's last method");**

**}**

**}**

**class B{**

**public synchronized void d2(A a) {**

**System.*out*.println("thread 2 started d2 mehod");**

**try {**

**Thread.*sleep*(1000);**

**} catch (Exception e) {**

**// TODO: handle exception**

**}**

**System.*out*.println("thraed 2 trying to call A's last method");**

**a.last();**

**}**

**public synchronized void last() {**

**System.*out*.println("inside B's last method");**

**}**

**}**

**class Mythread extends Thread{**

**A a = new A();**

**B b = new B();**

**public void m1() {**

**this.start();**

**a.d1(b);**

**}**

**@Override**

**public void run() {**

**b.d2(a);**

**}**

**}**

**public class TestClass{**

**public static void main(String[] args) {**

**Mythread t1 = new Mythread();**

**t1.m1();**

**}**

**}**

**Starvation:**

**It is situation where there is long waiting of thread ,but waiting will end at a certain point.**

**Green thread :**

**The thread which is compleately managed by jvm without taking any underlaying os support such thread is called green thread.**

**Thread group :**

**Based on functionality we can group threads in a single unit this is thread group**

**It can also contains sub thread group**

**Every thread group is child group of system group**

# Concurrent package :

**so in traditional synchronized keyword :**

**we are not having any flexiablity to try for a lock without waiting .**

**also there are some issues :**

**like way to specify maxium waiting time for a thread**

**or specify all the waiting threads**

**it may creats performance issues or deadlock like situations**

**to overcome these problem there is concurrent package in java which introduced in 1.5**

**Thread Safe :** Thread safety simply ensures that when a thread is modifying or reading shared data, no other thread can access it in a way that changes the data.

**Lock Interface :**

**It is similar to lock acquired by thread to execute synchronized method or block**

**It provides more operation than traditional synchronized lock**

**imp method of Lock interface :-**

**lock():**

**in this is if the lock is already available current thread will get the lock**

**if the lock is not already available then it will wait till getting the lock it is exactly as traditional synchronized method.**

**boolean trylock() :**

**to acquire the lock without waiting .**

**if the lock is available the thread acquires the lock and returns true**

**if lock is not available returns false and continue its execution without waiting**

**in this case thread is never going into waiting state**

**ex.**

**If(l.trylock){**

**Perform safe operation**

**-------------------**

**} else{**

**Perform alternative operation**

**}**

**Ex.**

**class Mythread extends Thread{**

**ReentrantLock l = new ReentrantLock();**

**public Mythread(String name) {**

**super(name);**

**}**

**@Override**

**public void run() {**

**if (l.tryLock()) {**

**System.*out*.println(Thread.*currentThread*().getName()+**

**" got lock and performing safe operation");**

**try {**

**Thread.*sleep*(2000);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**l.unlock();**

**}else {**

**System.*out*.println(Thread.*currentThread*().getName()+**

**" didn't get lock and performing alternative operation");**

**}**

**}**

**}**

**public class TestClass{**

**public static void main(String[] args) {**

**Mythread t1 = new Mythread("first thread ");**

**Mythread t2 = new Mythread("second thread");**

**t1.start();**

**t2.start();**

**}**

**}**

**tryLock(1000,timeunit.milisec/hr/min) :**

**if the lock is available the thread acquires lock**

**if lock is not available wait for some time still if you wan’t get lock**

**then continue other execution.**

**Ex.**

**class Mythread extends Thread{**

**ReentrantLock l = new ReentrantLock();**

**public Mythread(String name) {**

**super(name);**

**}**

**@Override**

**public void run() {**

**do {**

**try {**

**if (l.tryLock(2000, TimeUnit.*MILLISECONDS*)) {**

**System.*out*.println(Thread.*currentThread*().getName()+**

**" got lock");**

**Thread.*sleep*(1000);**

**l.unlock();**

**System.*out*.println(Thread.*currentThread*().getName()+**

**" releases lock");**

**break;**

**}else {**

**System.*out*.println(Thread.*currentThread*().getName()+**

**" unable to get lock trying....");**

**}**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**} while (true);**

**}**

**}**

**public class TestClass{**

**public static void main(String[] args) {**

**Mythread t1 = new Mythread("first thread ");**

**Mythread t2 = new Mythread("second thread");**

**t1.start();**

**t2.start();**

**}**

**}**

**lockInterreptibly() :**

**if lock is available immediately acquires the lock**

**if lock is not available it will wait**

**while waiting if the thread is interrupted it wont get the lock**

**if thread in not interrupted while waiting then only it will get the lock.**

**Unlock():**

**to unlock a lock**

**while calling this method current thread should be owner of the lock**

**otherwise we will get exception**

**ex.**

**l.lock();**

**---------**

**----------**

**l.unlock();**

**Reentrant lock class :**

**It is implementation class of lock interface**

**It is direct child class of object**

**Reentrant means thread can acquire same lock multipal times to makes strong hold on lock.**

**Internally jvm will maintans hold count**

**Lock will be released when hold count reaches zero**

**Ex.**

**l.lock();**

**l.lock();**

**l.lock(); threadCount = 3**

**………………………..**

**l.unlock();**

**l.unlock()**

**l.unlock() threadCount = 0**

**ReentrantLock l = new ReentrantLock();**

**No fair policy any thread can get chance**

**ReentrantLock l = new ReentrantLock(boolean true);**

**Applies fair policy longest waiting thread will get the chance**

**Methods :**

**Boolean isHeldbyCurrentThread() : lock hold by current thread or not**

**getQueudThreads(): it returns collection of threads which are waiting for a lock**

**class Display{**

**public void disp(String name) {**

**ReentrantLock rLock = new ReentrantLock();**

**rLock.lock();**

**for (int i = 0; i <5; i++) {**

**System.*out*.print("hello : ");**

**try {**

**Thread.*sleep*(1000);**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**System.*out*.println(name);**

**}**

**rLock.unlock();**

**}**

**}**

**class Mythread extends Thread{**

**Display display;**

**String name;**

**public Mythread(Display display, String name) {**

**super();**

**this.display = display;**

**this.name = name;**

**}**

**@Override**

**public void run() {**

**display.disp(name);**

**}**

**}**

**public class TestClass{**

**public static void main(String[] args) {**

**Display display = new Display();**

**Mythread t1 = new Mythread(display, "sid");**

**Mythread t2 = new Mythread(display, "kid");**

**t1.start();**

**t2.start();**

**}**

**}**

# Thread Pool/Executor Framework :-

**So if we creating thread for every new job this will cause performance and memory problem to overcome this we use executor framework.**

**It is pool of already created thread.**

**Introduced in java 1.5**

**While devlopeing and desigening web servers and application servers we use thread pool.**

**(i)ExecutorService service = Executor.newFixedThreadPool(size no.);**

**Service.Submit(job) : we can submit runnable job using this method**

**Service.shutdown() : we can shutdown ExecutorSevice by using this method**

**Uses :**

It is used in scenarios like sending the mails or transferring files asynchronously.

the major benefit of using the thread pool is that :

* it reduces the response time by avoiding thread creation during request or task processing
* Secondly, executor framework handle the thread management, so you don't need to take care of it.

**Ex.**

**class** PrintJob **implements** Runnable {

String name;

**public** PrintJob(String name) {

**this**.name = name;

}

@Override

**public** **void** run() {

System.***out***.println(name + " job started by " + Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println(name + " job ended by " + Thread.*currentThread*().getName());

}

}

**public** **class** Test101 {

**public** **static** **void** main(String[] args) {

PrintJob[] jobs = { **new** PrintJob("siddhes"),

**new** PrintJob("manoj"),

**new** PrintJob("manohar"),

**new** PrintJob("nira"),

**new** PrintJob("deepak"),

**new** PrintJob("ashok") };

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

**for** (PrintJob printJob : jobs) {

service.submit(printJob);

}

service.shutdown();

}

}

# Callable (I) :

**In case of runnable job thread it wont return anything**

**If a thread is required to return some result after execution**

**Then we should go for callable.**

**It contains Call () method**

**If we submit callable object to executor framework then after compleating the job thread returns object of type future ,it used to retrive result from callable job.**

**Ex.**

**class Mycallable implements Callable<Integer>{**

**int num;**

**public Mycallable(int num) {**

**this.num = num;**

**}**

**@Override**

**public Integer call() throws Exception {**

**System.*out*.println(Thread.*currentThread*().getName()+**

**" is responsible for sum of first "+num+" numbers");**

**int sum = 0;**

**for (int i = 0; i <= num; i++) {**

**sum = sum+i;**

**}**

**return sum;**

**}**

**}**

**public class TestClass{**

**public static void main(String[] args) {**

**Mycallable job[] = {new Mycallable(10),**

**new Mycallable(10),**

**new Mycallable(5),**

**new Mycallable(2),**

**new Mycallable(9),**

**new Mycallable(6)};**

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

**for (Mycallable mycallable : job) {**

**Future<Integer> future = service.submit(mycallable);**

**try {**

**System.*out*.println(future.get());**

**} catch (InterruptedException | ExecutionException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

# ThreadLocal :

**It provides thread local variables**

**It maintains value per thread bases**

**Every thread local maintains seprate values like userid,password,transation id etc…**

**Once thread enters into dead state all its local variable by default available for garbage collection.**

# Design Pattern

## **Design patterns are the reusable solutions that solve common problems of software development.**

creational design

“structural” design pattern

“behavioral” software design pattern

**Architectural Patterns**

## Factory design pattern :

It is a creational design pattern

it says that define an interface or a abstract class and let the subclasses decide which object to instantiate.

It basically means there is a method which is a factory, and as in general factories are involved with creational stuff and so like that here we are creating object. It is one of the best ways to create an object where object creation logic is hidden to the client.

Ex.

**abstract** **class** vehicle{

**public** **abstract** **int** getWheel();

@Override

**public** String toString() {

**return** "vehicle [getWheel()=" + getWheel() + "]";

}

}

**class** car **extends** vehicle{

**int** wheel;

**public** car(**int** wheel) {

**this**.wheel = wheel;

}

@Override

**public** **int** getWheel() {

**return** **this**.wheel;

}

}

**class** bike **extends** vehicle{

**int** wheel;

**public** bike(**int** wheel) {

**this**.wheel = wheel;

}

@Override

**public** **int** getWheel() {

**return** **this**.wheel;

}

}

**class** VehicleFactory{

**public** **static** vehicle getInstance(String type,**int** wheel) {

**if** (type=="car") {

**return** **new** car(wheel);

}

**if** (type == "bike") {

**return** **new** bike(wheel);

}

**return** **null**;

}

}

**public** **class** TestClass{

**public** **static** **void** main(String[] args) {

vehicle carVehicle = VehicleFactory.*getInstance*("car", 4);

System.***out***.println(carVehicle);

vehicle bikeVehicle = VehicleFactory.*getInstance*("bike", 2);

System.***out***.println(bikeVehicle);

}

}

## **Singleton Design Pattern:**

**It is creational design pattern**

**in this class must ensure that only a single instance should be created and a single object can be used by all other classes.**

**It is designed to save memory.**

**The object is not created on every request, instead, a single instance is reused.**

**Example: We can use this in Database connection. Where single instance of that connection is created and only through which We can perform database operations**

**Ex.**

**class** SingletonDemo{

**private** **static** SingletonDemo *instanceDemo*;

**private** SingletonDemo(){

}

**public** **static** SingletonDemo getInstanceDemo() {

**if** (*instanceDemo* == **null**) {

*instanceDemo* = **new** SingletonDemo();

}

**return** *instanceDemo*;

}

}

**public** **class** TestClass{

**public** **static** **void** main(String[] args) {

SingletonDemo s1 = SingletonDemo.*getInstanceDemo*();

System.***out***.println(s1.hashCode());

SingletonDemo s2 = SingletonDemo.*getInstanceDemo*();

System.***out***.println(s2.hashCode());

}

}

There is chance this will broke :

The code will create multiple instance of singleton class if called by more then one thred(multithreading enviroment)

If two threads simultaneously checks **if** (*instance* == **null**) then they both will create a new instance and singleton will break.

To avoid this :

We can make get instance method synchronized but it will effect performance

So we have double check locking pattern , where only critical section of code is locked :

**public** **static** SingletonDemo getInstanceDemo() {

**if** (*instanceDemo* == **null**) { // multiapl thread are checking

**synchronized** (SingletonDemo.**class**) {

**if** (*instanceDemo* == **null**) { // one can thred can able to check here at a single time

*instanceDemo* = **new** SingletonDemo();

}

}

}

**return** *instanceDemo*;

}

## Façade **Design Pattern:**

It hides the complexities of the system and provides an interface to the client from where the client can access the system.

it provides a unified interface to a set of interface(s) in a subsystem.

It also defines a higher-level interface that makes the subsystem easier to use.

is a “structural” design pattern

The **Facade design pattern** is chosen because it:

* Simplifies complex systems by providing a simple interface.
* Improves code readability.
* Reduces coupling between clients and subsystems.
* Offers a cleaner, unified interface.
* Makes testing and maintenance easier.

When to use:

* To simplify interactions with complex systems.
* To reduce dependencies between clients and subsystems.

## The strategy pattern:

is a “behavioral” software design pattern.

It is useful for situations where it is necessary to dynamically swap the algorithms used in an application.

It define a family of algorithms, encapsulate each one as an object, and make them interchangeable.

The strategy pattern basically lets the algorithms vary independently from clients to client.

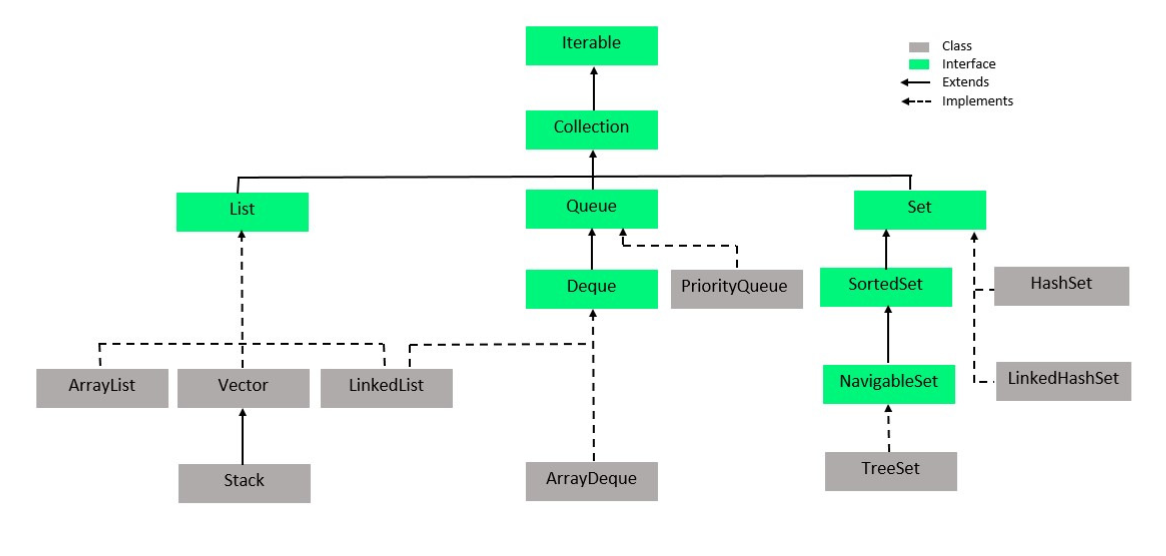
**Ioc design pattern**

**MVC design pattern**

## **Dao design pattern :**

Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business logic. Following are the participants in Data Access Object Pattern

# Collection

****

**If we want to represent group of individual object as a single entity then we go for collection**

**Methods:**

**Add(obj o) ,addAll(collections c), remove(obj o),isEmpty(),size(),c.toArray()**

**Clear() : clear the full collection 0 elements left .**

**List Interface :**

**direct child interface to Collection interface**

**its in JDK1.2 version**

**index based**

**able to allow duplicate elements.**

**followes insertion order.**

**not following Sorting order.**

**allow any number of null values.**

**allow heterogeneous elements**

**indexOf(obj o),lastIndexOf(obj 0),get(obj o)**

**set(int index,obj o) : replace the value**

## **Array vs arrayList**

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| **It is fixed length**  **Cannot change length of array once created**  **Can contains both primitives and objects** | **It is variable length collection**  **It is dynamic in length**  **Only store objects** |

## **Arraylist vs linklist**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedLIst** |
| **Introduced 1.2 version.**  **It is a direct implementation class to List interface.**  **Its internal data structure is "Resizable Array".**  **Its initial capacity is 10 elements.**  **It is best option for frequent retrieval operations.**  **It is not synchronized.**  **It will improve application performance.**  **It is not thread safe.**  **It is not Legacy Collection.**  **Constructors:**  **public ArrayList()**  **public ArrayList(int capacity)**  **public ArrayList(Collection c)**  It can be used to create an ArrayList object with all the elements of the specified Collection object | **Arraylist features +**  **internal data structure is "Double LinkedList"**  **Best for frequent deletion operation.**  **It is not synchronized Collection.**  **No method is synchronized in LinkedList.**  **public LinkedList(Collection c)** |

## **Vector(I) :**

**It was introduced in JDK1.0 version.**

**It is Legacy Collection.**

**It is a direct implementation class to List interface.**

**It is index based.**

**It allows duplicate elements.**

**It follows insertion order.**

**It will not follow sorting order.**

**It allows any number of null values.**

**Its internal data structure is "Resizable Array".**

**Its initial capacity is 10 elements.**

**It is best choice for frequent retrieval operations.**

**All the methods of vector class are synchronized.**

**It allows only one thread at a time.**

**It follows sequential execution.**

**It will increase execution time.**

**It will reduce application performance.**

**It is threadsafe.**

**Mehods () :**

**addElements(obj o), removeElements(obj o), removeElementsAt(int index),**

**removeAllElements();**

### **Statck (c):**

It was introduced in JDK 1.0 version,

it is a Legacy Collection

child class to Vector class.

It able to arrange all the elements as per "Last In First Out" [LIFO] alg.

Methods :

Push() , pop(), empty(), search()

Peak() : to return top of the stack

## Cousers :

if we want to retrive elements one by one

**Enumeration:**

**It is a Legacy Cursor, it is applicable for only Legacy Collections to retrieve**

**Ex .**

**Enumeration e=v.elements();**

**while(e.hasMoreElements())**

{

**System.out.println(e.nextElement());**

}

### Iterator :

Applicable for all the the collection items

Ex.

**Iterator it=list.iterator();**

**while(it.hasNext())**

{

**String element=(String)it.next();**

**System.out.println(element);**

**if(element.equals("C"))**

{

it.remove();

}

}

Size check is one of major difference between ForEach and Iterator

Iterator v/s foreach

While using foreach we can not modify collection items .

ListIterator :

**It allow to read elements in both forward direction and backward direction.**

**It able to allow the operations like read, insert, replace and remove**

**It is only applicable List Interface objects**

**Ex.**

**ListIterator lit=ll.listIterator();**

**while(lit.hasNext())**

16) {

**String element=(String)lit.next();**

**if(element.equals("B"))**

19) {

20) lit.add("X");

21) }

22) if(element.equals("D"))

23) {

24) lit.set("Y");

25) }

26) if(element.equals("E"))

27) {

28) lit.remove();

29) }

}

## Set (I) :

### Hashset vs linkedhashset

|  |  |
| --- | --- |
| HashSet | Linked HashSet |
| **implementation class to Set interface.**  **introduced in JDK 1.2 version.**  **arrange all the elements on the basis of hashcode.**  **not allow duplicate elements.**  **not follow insertion order.**  **not follow Sorting order.**  **It able to allow only one null value.**  **Its interal data structer is "Hashtable".**  **Its initial capacity is "16" elements and its initial fill\_Ratio is 75%.**  **Almost all the methods are not synchronized in HashSet**  **public HashSet(int capacity)**  HashSet hs = new HashSet(20);  **public HashSet(int capacity, float fill\_Ratio)**  HashSet hs = new HashSet(20, 0.85f);  **public HashSet(Collection c)** | introduced in 1.4 v  follow insertion order  internal data structer of LinkedHashSet is "Hashtable" and "LinkedList". |

### **SortedSet(I):**

**introduced in JDK1.2 version.**

**child interface to Set interface.**

**not allow duplicate elements.**

**not follow insertion order.**

**It follows Sorting order.**

**It only allows homogeneous elements.**

**not allow heterogeneous elements.**

**not allow null values.**

**It able to allow only Comparable objects by default.**

Note: If we are trying to add non comparable objects then we have to use Comparator.

**Methods:**

**public Object first()**

**public Object last()**

**public SortedSet headSet(Object obj)**

**public SortedSet tailSet(Object obj)**

**public SortedSet subSet(Object obj1, Object obj2)**

It will return elements

which are greater than or equals to the first element and which are less than the second element.

### **NavigableSet :**

It was introduced in JAVA 6 version

child interface to SortedSet interface

it has methods to provide navigations over the elements.

**Methods: use these methods using TreeSet class**

**public Object ceiling(Object obj)**

It will return lowest element among all the elements which are greater than or

Equals to the specified element.

**public Object higher(Object obj)**

It will return lowest element among all the elements which are greater than the specified element.

**public Object floor(Object obj)**

It will return highest element among all the elements which are less than or equals to the specified element.

**Trpublic Object lower(Object obj)**

It will return highest element among all the elements which are less than the specified element.

**public Object pollFirst()**

It will remove and return first element from NavigableSet.

**public Object pollLast()**

It will remove and return last element from NavigableSet.

**public NavigableSet descendingSet()**

It will return all elements in the form of NavigableSet in descending order.

### TreeSet (c):

**introduced in JDK 1.2 version.**

**It has provided implementation for Collection, Set, SortedSet and NavigableSet interfaces.**

**It is not allowing duplicate elements.**

**It is not following insertion order.**

**It follows Sorting order.**

**It allows only homogeneous elements.**

**It will not allow null values**

**Its internal data structure is "Balanced Tree".**

**It is mainly for frequent search operations.**

**It store elements by default natural sorting order using compariable internally.**

**Ex**

**public class TestClass{**

**public static void main(String[] args) {**

**Comparator<Integer> comparator = new Comparator<Integer>() {**

**@Override**

**public int compare(Integer o1, Integer o2) {**

**if (o1 < o2) {**

**return 1;**

**}**

**else if (o1>o2) {**

**return -1;**

**}**

**else {**

**return 0;**

**}**

**}**

**};**

**TreeSet<Integer> treeSet = new TreeSet<>(comparator);**

**treeSet.add(45);**

**treeSet.add(5);**

**treeSet.add(4);**

**treeSet.add(54);**

**System.*out*.println(treeSet);**

**}**

**}**

## Compariable Interface : -

It is used to comare object with itself

It used to sort user define object

Internally it uses compareTo() method

Ex.

**class laptop implements Comparable<laptop>{**

**private String brand;**

**private int price;**

**private int code;**

**// parameterize constructor**

**//getters setters**

**// toString**

**@Override**

**public int compareTo(laptop o) {**

**if (this.code > o.code) {**

**return 1;**

**}**

**else if (this.code < o.code) {**

**return-1;**

**}**

**else {**

**return 0;**

**}**

**}**

}

**public class TestClass{**

**public static void main(String[] args) {**

**List<laptop> laptops = new ArrayList<>();**

**laptops.add(new laptop("dell", 800, 45180));**

**laptops.add(new laptop("hp", 700, 146454));**

**laptops.add(new laptop("lenovo", 750,565180));**

**laptops.add(new laptop("apple",1000 , 4454180));**

**Collections.*sort*(laptops);**

**for (laptop laptop : laptops) {**

**System.*out*.println(laptop);**

**}**

**}**

}

## Comparator Interface:

When class in not implementing comparable we use comparator.

It also used when we want to change the way of sorting value.

It contains inner class in that we can specify sorting logic.

Ex.

**class laptop{**

**private String brand;**

**private int price;**

**private int code;**

**//parametarize constructor**

**//setters getters**

**//toString**

**}**

**public class TestClass{**

**public static void main(String[] args) {**

**List<laptop> laptops = new ArrayList<>();**

**laptops.add(new laptop("dell", 800, 45180));**

**laptops.add(new laptop("hp", 700, 146454));**

**laptops.add(new laptop("lenovo", 750,565180));**

**laptops.add(new laptop("apple",1000 , 4454180));**

**for (laptop laptop : laptops) {**

**System.*out*.println(laptop);**

**}**

**Comparator<laptop> comparator = new Comparator<laptop>() {**

**@Override**

**public int compare(laptop o1, laptop o2) {**

**if (o1.getPrice() < o2.getPrice()) {**

**return 1;**

**}**

**else if (o1.getPrice() > o2.getPrice()) {**

**return -1;**

**}**

**else {**

**return 0;**

**}**

**}**

**};**

**Collections.*sort*(laptops,comparator);**

**System.*out*.println("after : ");**

**for (laptop laptop : laptops) {**

**System.*out*.println(laptop);**

**}**

**}**

**}**

## **Queue (I) :**

**It used to manage all elements prior to processing.**

**It was introduced in JDK 5.0 version.**

**It able to arrange all the elements as per FIFO [First In First Out], but, it is possible to change this algorithm as per our requirement.**

**allow duplicate elements.**

**not following Insertion order.**

**It is following Sorting order.**

**not allow null values**

**It able to allow only homogeneous elements.**

**It able to allow comparable objects by default.**

**Linked list is also a implementation class of Queue**

**Methods:**

**public void offer(Object obj) : to add elements**

**public Object peek() : return head element/null**

**public Object element() : return head element /excepetion**

**public Object poll() : remove and return head element**

**public Object remove() : remove element**

**ProprityQueue(c) :**

**if we want to represent group of object prior to processing according to some priority**

**implementation class of queue**

**priority can be default natural sorting order**

**or it can be customize sorting order by using compareter**

**insertion order not preserved**

**duplicates not allowed**

**null is not allowed**

**not thread safe**

**initial capacity is 11**

**constructor :**

**public PriorityQueue(int capacity,Comparator c)**

**deque (i):**

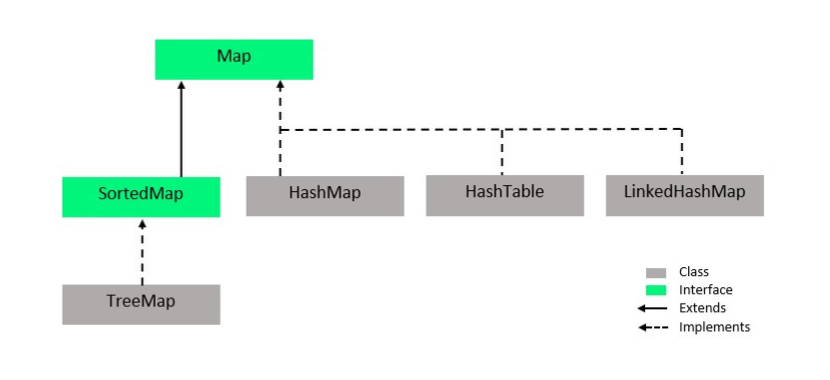
**implementation of queue**

**it is also known as double ended queue**

**insertion and deletion operation can be possible for both of ends.**

**Can access elements using iterator**

# Map(i)

****

**If we want to represent group of objects as a key value pair then we should go for Map interface.**

**introduced in JDK 1.2 version.**

**It is not child interface to Collection Interface.**

**It able to arrange all the elements in the form of Key-value pairs.**

**In Map, both keys and values are objects.**

**Duplicates keys are not allowed, but values can be duplicated.**

**Only one null value is allowed at keys side, but, any number of null values are allowed at values side.**

**Both keys and Values are able to allow heterogeneous elements.**

**Insertion order is not followed.**

**Sorting order is not followed.**

## Itereate map :

Map<String, String> map = new Hashmap<>();

1)

for (map.Entry<String, String> entry : map.entrySet()) {

System.out.println(entry.getKey() + "/" + entry.getValue());

}

2)

**Methods:**

**public void put(Object key, Object value)**

**public void putAll(Map m)**

public Object get(Object key) : It will return value of the specified key

**public Object remove(Object key)**

**public int size()**

public boolean containsKey(Object key) : check whether the specified key is existed or not

**public boolean cotainsValue(Object key) : check whether the specified value is existed or not**

**public Set keySet() : return all the keys in set form**

**public Collection values()**

**public boolean isEmpty()**

## **HashMap(c):**

**It was introduced in JDK1.2 version.**

**It able to arrange all the elements in the form of Key-value pairs.**

**Duplicates are not allowed at keys, but values may be duplicated.**

**Only one null value is allowed at keys side, but, any number of null values are allowed**

**at values side.**

**Insertion order is not followed.**

**Sorting order is not followed.**

**Its internal data structure is "Hashtable".**

**Its initial capacity is 16 elements.**

**No method is synchronized in HashMap**

**It is not threadsafe.**

**Constructor :**

**public HashMap(int capacity, float fill\_Ratio)**

**LinkedHashMap(c):**

**LinkedHashMap was introduced in JDK 1.4 version.**

**LinkedHashMap is following insertion order.**

LinkedHashMap internal data structure is Hashtable+LinkedList

**IdentityHashMap(c):**

**IdentityHashMap was introduced in JDK 1.4 version.**

**to check duplicate keys HashMap will use equals(-) method, but, IdentityHashMap will use '==' operator.**

**WeakHashMap(c):**

**Once if we add an element to HashMap then HashMap is not allowing Garbage Collector to destroy its objects.**

**Even if we add an element to WeakHashMap then WeakHashMap is able to allow Garbage Collector to destroy elements.**

### **SortedMap(i):**

**introduced in JDK1.2 version.**

**child interface to Map interface**

**It will not allow duplicate elements at keys side**

**but it will allow duplicate elements at value side.**

**It will not follow insertion order.**

**It will follow sorting order.**

**It will not allow null values at keys side**

**It will not allow heterogeneous elements at keys side.**

**Methods:**

public Object firstKey() public Object lastKey()

public SortedMap headMap(Object key) public SportedMap tailMap(Object key)

public SortedMap subMap(Object obj1, Object obj2)

**NavigableMap(i):**

It was introduced in JAVA6 version

it is a child interface to SortedMap

it has defined methods to provide navigations over the elements.

**Methods:**

**public Object CeilingKey(Object obj)**

**public Object higherKey(Object obj)**

**public Object floorKey(Object obj)**

**public Object lowerKey(Object obj)**

**public NavigableMap descendingMap()**

**public Map.Entry pollFirstEntry()**

**public Map.Entry pollLastEntry()**

### **TreeMap(c):**

**It was introduced in JDK 1.2 version.**

**implementation class to Map, SortedMap and NavigableMap interfaces.**

**It will not allow duplicate elements at keys side, But values can be duplicate**

**It will not follow insertion order.**

**It will follow sorting order.**

**It will not allow null values at keys side**

**It will not allow heterogeneous elements at keys side**

**Its internal data Structer is "Red-Black Tree".**

**No methods are synchronized in TreeMap.**

**It is not threadsafe.**

**Constructors:**

**public TreeMap()**

**public TreeMap(Comparator c)**

**public TreeMap(Map m)**

### **hashmap vs hashtable**

|  |  |
| --- | --- |
| **Hashmap** | **Hashtable** |
| **introduced in JDK 1.2 version.**  **HashMap is not Legacy class**  **one null value is allowed at keys side**  **any null values are allowed at value side**  **is not synchronized.**  **Not thread safe** | **introduced in JDK 1.0 version.**  **HashMap is Legacy class.**  **null value not allowed at keys side**  **null values not allowed at value side also**  **it is synchronized.**  **Thread safe** |

## **Internal working of Hashmap and hashSet :**

**Internal data structure of hashmap or hashset is hashtable.**

**When we create hashmap object and and call .put method internally it will create 16 buckets(tables) from (0-15),for these buckets threshold value is 12.**

**Internally each bucket maintain linked-list in form of node, where each node contains hash|key|value|next**

**When we add key and values in .put(k,v) internally it will calculate hashcode based on key**

**Then we further perform operation on that hashcode [index = hash&(n-1)] and it will generate us index value and on that index we store our values in form of node**

**Case 1 : if hash collides then it will checks by equals if both keys are equals or not if it not equals then it will will put that node next to existing node.**

**Case 2 : if keys are also same , then it will replace values**

**Case 3 : if key is null it will store it in 0th bucket**

**When we try to get values from map it will get complex as its buckets internally using linked-list, data retriving will cause performance drop , to overcome this java8 has enhancement :**

**Java 8 hashmap replaces linked-list with binary tree when number of elements in bucket reaches certain threshold**

**While converting list to binary tree, hashcode use for branching if there are two different hashcode in same bucket one is considerd smaller and another is larger**

**But when Both hascodes are equal hashcode assumes that key is comparable**

**And compares the key to determine direction so that some order can be maintained,it good practice to make the keys of hashmap comparable.**

# Concurrent Collection :

**Why concurrent collections :**

**Traditional collection objects are not thread safe, more then one thread can access them at a single time**

**There are some thread safe collection but there performance is very slow**

**In those thread safe collections for read operation also one thread is allowed at a time .**

**Because of this it will increase waiting time for other thread**

**ConcurrentModificationException :**

**It is big problem with traditional collection**

**When one thread is iterating over collection another thread is not allowed to modify that collection.**

**Hence these collections are not sutiable for multithreaded environment , to overcome this we have concurrent collections.**

## **ConcurrentMap(I) :**

**object putIfAbsent(obj key,obj value) :**

**add entry to map only if specified key is not already available.**

**Boolean remove(obj key,obj value):**

**Only remove if the key matches with specified value**

**Boolean replace(obj key,obj oldValue,obj NewValue) :**

**Only replace if the key matches with specified value**

## **ConcurrentHashMap(C) :**

**Underlaying data structure is hashtable.**

**It allows read and update thread safe operation concurrently**

**For read operation thread won’t require any lock where for update operation thread require lock but only specific part of lock(bucket level lock).**

**Null is not allowed for both key and value**

**It can perform update operation while itereating.**

**It allow 16 update operation while reading.**

**Ex1.**

**public** **class** TestClass{

**public** **static** **void** main(String[] args) {

ConcurrentHashMap<String, String> map = **new** ConcurrentHashMap<>();

map.put("s1","siddhesh");

map.put("s2", "manoj");

map.putIfAbsent("s1", "aakash");

map.putIfAbsent("s3", "nilam");

map.remove("s1", "manoj");

map.replace("s3", "nilam", "abhi");

System.***out***.println(map);

}

}

**Ex2.**

**public** **class** TestClass **extends** Thread{

**static** ConcurrentHashMap<String, String> *map* = **new** ConcurrentHashMap<String, String>();

@Override

**public** **void** run() {

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

}

System.***out***.println("updating...");

*map*.put("s3", "navin");

}

**public** **static** **void** main(String[] args) **throws** InterruptedException {

*map*.put("s1", "siddhesh");

*map*.put("s2", "manoj");

TestClass testClass = **new** TestClass();

testClass.start();

KeySetView<String, String> set = *map*.keySet();

Iterator<String> iterator = set.iterator();

**while** (iterator.hasNext()) {

System.***out***.println("iterating...");

String string = (String) iterator.next();

System.***out***.println(string);

Thread.*sleep*(1000);

}

System.***out***.println(*map*);

}

}

## **Hashmap vs concurrent hashmap**

|  |  |
| --- | --- |
| **Hashmap** | **Concurrent HAshmap** |
| It is not thread safe  Performance is high  Iterator is fail-fast is throws concurrentModificationException.  Null is allowed for both key and value  Introduced in 1.2 | It is thread safe  Performance is relatively low  Iterator is fail-safe it won’t throw any exception  Null not allowed for both  Introduced in 1.5 |

**CopyOnArrayList :**

**It is thread safe version of arraylist**

**It create clone of underlaying arraylist for every write operation, at certain point both the arraylist gets synchronized jvm takes care of that.**

**It is fail-safe.**

**addIfAbsent(obj o)**

**addAllAbsent(collection c)**

**CopyOnWriteArraySet :**

**it is thread safe version of set**

**internally implemented by copy on write array list**

**insertion order is preserved**

**duplicates not allowed**

**It create clone of underlaying arraylist for every write operation, it can perform multiple read operations at a single time**

**It is costly ,not to use for large number of write operations**

**It can not perform read and remove operation at a same time.**

# Java I/O:

## **FileWriter:**

This character-oriented Stream can be used to transfer the data from Java Application to a particular target File.

Ex.

FileWriter fw = new FileWriter("abc.txt",true);

## **FileReader:**

This character-oriented stream can be used to transfer the data from a particular source file to Java program.

Ex.

# Java 8

## **Java 8 New Features:**

**Lambda Expression**

**Functional Interfaces**

**Default methods**

**Predicates**

**Functions**

**Double colon operator (::)**

**Stream API**

**Date and Time API**

## **Lambda Expression :**

**It used to enable functional programming in java**

**To write readable ,maintainable more consies code**

**It as an anonymous function**

**Without return type function**

**Without modifier**

**Without any name**

**For lambda expression no .class file generated**

**Ex.**

**()->{sysout(“hello world”);}**

**Imp Rules of lambda :**

**If the body of lambda expression contains only one statement then Curly-braces are optional.**

**Complier will automatically refer the type of a variable passed in arguments hence type is also optional**

## **Functional Interface :**

**It is a interface which contains only single abstract method**

**Job of function interface to call lambda expression.**

**Functional interface should contain only one abstract method**

**But there can be many solid methods are there.**

**If an interface extends Functional Interface and child interface doesn’t contain any abstract method then child interface is also Functional Interface**

**In the child interface we can’t define any new abstract methods otherwise child interface won’t be Functional Interface**

**Ex.**

**interface Interface2{**

**public abstract int length(String str);**

**}**

**public class TestClass{**

**public static void main(String[] args) {**

**\* Interface2 interface2 = (s)->{**

**int temp = s.length();**

**return temp \*2;**

**};**

**System.*out*.println(interface2.length("siddh"));**

**\* Runnable runnable = ()->{**

**for( int i = 1;i<10;i++) {**

**System.*out*.println("calling run - "+i);**

**Thread.*sleep*(1000);**

**}**

**System.*out*.println("operation ended");**

**};**

**Thread t1 = new Thread(runnable);**

**t1.start();**

**}**

**}**

## **Anonymous inner class :**

**it is a class without any name or return type**

**Anonymous inner class can extend concrete class**

**can extend abstract class**

**can implement interface with any number of methods**

**ex. Comparator ,runnable ,callable**

## **Anonymous Inner class v/s Lambda Expression**

|  |  |
| --- | --- |
| **Anonymous Inner class** | **Lambda Expression** |
| **It’s a class without name** | **It’s a method without name (anonymous function)** |
| **Anonymous inner class can extend**  **Abstract and concrete classes** | **lambda expression can’t extend**  **Abstract and concrete classes** |
| **can implement An interface that contains any number of**  **Abstract methods** | **can implement an**  **Interface which contains single abstract method (Functional Interface)** |
| **Inside anonymous inner class we can Declare instance variables.** | **Inside lambda expression we can’t**  **Declare instance variables, whatever the variables declared are simply acts as local variables.** |
| **Inside anonymous inner class “this” Always refers current anonymous Inner class object but not outer class Object.** | **Inside lambda expression “this”**  **Always refers current outer class object. That is enclosing class object.** |
| **Anonymous inner class is the best choice If we want to handle multiple methods.** | **Lambda expression is the best Choice if we want to handle interface**  **With single abstract method** |
| **At the time of compilation a separate**  **Dot class file will be generated (outerclass$1.class)** | **At the time of compilation no dot Class file will be generated for Lambda expression. It simply converts in to private method outer class.** |

## **Default Method :**

**Until 1.7 every method in interface is always public and abstract**

**After 1.8 default methods are also allowed in interface**

**Without effecting implementing classes if we have to add a new method to a interface then we use default method**

**It provides protection to implementing classes**

**Default methods are not allowed inside class**

**Every variable declared inside interface is always public static final whether we are declaring or not.**

**Note: We can’t override object class methods as default methods inside interface otherwise we get compile time error.**

**Default method vs multiple inheritance**

**Two interfaces can contain default method with same signature then there may be a chance of ambiguity problem (diamond problem) to the implementation class.**

**To overcome this problem compulsory we should override default method in the implementation class**

**Ex.**

**interface phone{**

**default void call() {**

**System.*out*.println("calling to someone from phone");**

**}**

**}**

**interface Telephone{**

**default void call() {**

**System.*out*.println("calling to someone from telephone");**

**}**

**}**

**class Device implements phone,Telephone{**

**@Override**

**public void call() {**

**System.*out*.println("only Calling..");**

**}**

**}**

**public class TestClass extends Device{**

**public static void main(String[] args) {**

**TestClass testClass = new TestClass();**

**testClass.call();**

**}**

**}**

## **Differences between interface with default methods and abstract class**

**Even though we can add concrete methods in the form of default methods to the interface, it won’t be equal to abstract class.**

|  |  |
| --- | --- |
| **Interface with Default Methods** | **Abstract Class** |
| **Inside interface every variable is Always public static final and there is No chance of instance variables** | **Inside abstract class there may be a Chance of instance variables**  **which Are required to the child class.** |
| **Interface never talks about state of Object.** | **Abstract class can talk about state of Object.** |
| **Inside interface we can’t declare**  **Constructors.** | **Inside abstract class we can declare**  **Constructors.** |
| **Inside interface we can’t declare**  **Instance and static blocks.** | **Inside abstract class we can declare**  **Instance and static blocks.** |
| **Functional interface with default**  **Methods Can refer lambda expression.** | **Abstract class can’t refer lambda**  **Expressions.** |
| **Inside interface we can’t override Object class methods.** | **Inside abstract class we can override Object class methods.** |

## **Static Method :**

**From 1.8 version onwards in addition to default methods we can write static methods also inside interface to define utility functions.**

**Interface static methods by-default not available to the implementation classes hence by using implementation class reference we can’t call interface static**

**We can also write main method in interface from 1.8**

**We call these methods by interface name**

**Ex.**

**interface Display{**

**static void disp() {**

**System.*out*.println("Disp method...");**

**}**

**}**

**public class TestClass implements Display{**

**public static void main(String[] args) {**

**Display.*disp*();**

**}**

**}**

## **Predefine interface :**

### **1)Predicate :**

**It is a function with a single argument and returns boolean value.**

**It’s a functional interface and it contains only one method : test()**

**Ex**

**public class TestClass{**

**public static void main(String[] args) {**

**for (int i = 0; i < 20; i++) {**

**Predicate<Integer> predicate = e -> e%2==0;**

**if (predicate.test(i)) {**

**System.*out*.print(i+" ");**

**}**

**}**

**}**

**}**

**Predicate joining :**

**Ex**

**public class TestClass{**

**public static void main(String[] args) {**

**for (int i = 0; i < 10; i++) {**

**Predicate<Integer> predicate = e -> e>2;**

**Predicate<Integer> predicate2 = e-> e<5;**

**if (predicate.and(predicate2).test(i)) {**

**System.*out*.print(i+" ");**

**}**

**if (predicate.or(predicate2).test(i)) {**

**System.*out*.print(i+" ");**

**}**

**if (predicate2.negate().test(i)) {**

**System.*out*.print(i+" ");**

**}**

**}**

**}**

**}**

### **2) function :**

**It is same as predicates except that functions can return any type of result**

**function can return only one value and that value can be any type.**

**It contains only one method : apply()**

**Ex.**

**//calculate square**

**public class TestClass{**

**public static void main(String[] args) {**

**Function< Integer, Integer> function = e->e\*e;**

**for (int i = 0; i < 10; i++) {**

**System.*out*.println(i+" : " +function.apply(i));**

**}**

**}**

**}**

**Function chaining :**

**public class TestClass{**

**public static void main(String[] args) {**

**for (int i = 1; i <= 10; i++) {**

**Function<Integer, Integer> function = e->e\*2;**

**Function<Integer, Integer> function2 = e->e\*e;**

**System.*out*.println(function.andThen(function2).apply(i));**

**}**

**}**

**}**

### **4) Consumer :**

**It takes input perform operation but won’t return anything**

**It contains accept() method**

**Ex.**

**public class TestClass{**

**public static void main(String[] args) {**

**Consumer<String>consumer = e->{System.*out*.println(e);};**

**String arr[] = {"siddhesh","maoj","madiwale"};**

**for (String string : arr) {**

**consumer.accept(string);**

**}**

**}**

**}**

### **5) Supplier :**

**It won’t take any input only provide output**

**It contains get() method**

**Ex.**

**//genrate otp**

**public class TestClass{**

**public static void main(String[] args) {**

**Supplier<String>supplier = ()->{**

**String otp = "";**

**for(int i = 0;i<=5;i++) {**

**otp = otp+((int)(Math.*random*()\*10));**

**}**

**return otp;**

**};**

**System.*out*.println("OTP : "+supplier.get());**

**}**

**}**

**Two argument Functional Interface :**

**Bipredicate**

**Biconsumer**

**Bifunction**

**Premetive Version of functional interface :**

**To avoid conversion we use this**

**Intpredicate**

**Doublepredicate**

**longPredicate**

**Doublefunction**

**If the input and output args are same then we should go for UnaryOperator <>.**

**If the two input and output args are same then we should go for BinaryOperator <>.**

## **Optional class:**

it is like container object that used to contain not-null objects.

it is used to represent null with absent value.

Like optional class provide us some utility methods to handle values as ‘available’ or ‘not available’ instead of checking null values.

.get() .isPresent()

## **Method Reference :**

**These are alternatives to lambda expression.**

**We can use this using double colan ::**

**We can use method refernce with functional interfaces**

**With the help of this we can provide reference of already existing method which improves code reusability .**

**In case of method reference both method args type and num of args must be match**

**Ex.**

**public class TestClass{**

**public static void m1() {**

**for (int i = 0; i <= 5; i++) {**

**System.*out*.println("child thread");**

**}**

**}**

**public static void main(String[] args) {**

**Runnable runnable = TestClass::*m1*;**

**Thread thread = new Thread(runnable);**

**thread.start();**

**System.*out*.println("started by main thread");**

**}**

**}**

## **Constructor References**

**Java 8 has feature so We can use :: ( double colon )operator to refer constructor**

**Ex.**

**class Student{**

**public Student() {**

**System.*out*.println("student class constructor");**

**}**

**}**

**interface Subject{**

**public Student math();**

**}**

**public class TestClass{**

**public static void main(String[] args) {**

**Subject subject = Student::new;**

**Student s1 = subject.math();**

**}**

**}**

**In above example ,**

**Subject S = Student :: new;**

**functional interface s referring student class constructor**

## **Streams :**

**If we want to process objects from collection the we use streams**

**it contains stream() method**

**filter() – to filter out collection object**

**collect() – to collect filter out object**

**map() – for every object if you want to perform some operation then we go for map()**

**inbuilt Methods :**

**count ()**

**sorted()**

**forEach(Consumer)**

**ToArray()**

**Stream.of() : We can also apply a stream for group of values and for arrays.**

### **Intermidate v/s terminal :**

|  |  |
| --- | --- |
| **Intermidate** | **Termainal** |
| **The operator which returns another stream as a result is called**  **Ex. filter,map,flatmap,sort** | **The operator which returns non stream value like collections objects or returns nothing called**  **Ex. forEach, .collect(Collectors.Tolist)** |

**Ex.**

**Stream s=Stream.of(99,999,9999,99999);**

**s.forEach(System.out:: println);**

**Double[] d={10.0,10.1,10.2,10.3};**

**Stream s1=Stream.of(d); s1.forEach(System.out :: println);**

**ex.**

**public class TestClass{**

**public static void main(String[] args) {**

**ArrayList<String> list = new ArrayList<>();**

**list.add("sid");**

**list.add("manoj");**

**list.add("madiwale");**

**List<String> list2 =**

**list.stream().filter(i->i.length()>5).collect(Collectors.*toList*());**

**System.*out*.println(list2);**

**}**

**}**

### **even numbers exist in the list ?**

**public class JavaHungry {**

**public static void main(String args[]) {**

**List<Integer> myList = Arrays.asList(10,15,8,49,25,98,32);**

**myList.stream()**

**.filter(n -> n%2 == 0)**

**.forEach(System.out::println);**

**}**

**}**

### **repeating numbers in stream**

**public** **class** TestClass{

**public** **static** **void** main(String[] args) {

List<Character> list = Arrays.*asList*('a','b','b','c','d','a');

Set<Character> set = list.stream()

.filter(e->Collections.*frequency*(list,e)>1)

.collect(Collectors.*toSet*());

System.***out***.println(set);

}

}

### sum of integer in list

List<Integer> list = Arrays.*asList*(22,3,9,4,76);

Optional<Integer> sum = list.stream().reduce((a,b)->a+b);

System.***out***.println("sum : "+sum.get());

**Reduce():**it is usedhandle two parameters, where the first parameter is the previous return value,and the second parameter is the current calculate value in the stream, so in this way it reduce to single result.

### calculate average of all the numbers given

**double** asDouble = list.stream().mapToInt(e-> e).average().getAsDouble();

//get the squre

**double** result = list.stream().map(e->e\*e)

### //check if it is greater then 100

.filter(e->e>100)

### //convert element to integer

.mapToInt(e->e)

### //calculate avarage

.average()

### //convert that in double

.getAsDouble();

### **odd and even numbers from given list**

List<Integer> even =

list.stream().filter(e->e%2==0).collect(Collectors.*toList*());

List<Integer> odd =

list.stream().filter(e->e%2!=0).collect(Collectors.*toList*());

### **numbers starting with 2**

List<Integer> numList =

//convert to string

list.stream().map(e->String.*valueOf*(e))

//check if starts with 2

.filter(e->e.startsWith("2"))

//convert to integer

.map(e-> Integer.*valueOf*(e))

.collect(Collectors.*toList*());

### **find max and minimum number in given list**

Integer maxInteger =

list.stream()

.max(Comparator.*comparing*(e->Integer.*valueOf*(e))).get();

Integer minInteger =

list.stream()

.min(Comparator.*comparing*(e->Integer.*valueOf*(e))).get();

### **sort list in asc and dec order**

List<Integer> AscList = list.stream().sorted().collect(Collectors.*toList*());

List<Integer> DecList =

list.stream().sorted(Collections.*reverseOrder*()).collect(Collectors.*toList*());

### **get first 5 numbers**

List<Integer> getFirst5 = list.stream().limit(5).collect(Collectors.*toList*());

### skip first 5 numbers

List<Integer> skipFirst5 = list.stream().skip(5).collect(Collectors.*toList*());

### **secondHigest and SecondLowest num from** list?

Integer secondHighest =

list.stream().sorted(Collections.*reverseOrder*())

.distinct().skip(1).findFirst().get();

Integer secondLowest = list.stream().sorted().distinct().skip(1).findFirst().get();

### convert list to map

Map<String, Integer> collect =

list.stream()

.collect(Collectors.*toMap*(vender::getIDString, vender::getLocationZipNo));

//for duplicate key approach

Map<String, List<Integer>> map =

list.stream()

.collect(

Collectors.

*groupingBy*(vender::getIDString,

Collectors.

*mapping*(vender::getLocationZipNo),Collectors.*toList*())

)

);

Grouping by : it is used to group object by specified property

Mapping :  It converts a Collector accepting elements of one type to other type

### **convert map to list**

List<String> list = new ArrayList<String>(m.values());

## **Q.Array to list**

**int** a[] = {1,2,3};

IntStream stream = Arrays.*stream*(a);

List<Integer> collect = Arrays.*stream*(a).boxed().collect(Collectors.*toList*());

//or

List<Integer> list2 = **new** ArrayList<>(a.length);

**for** (**int** i : a) {

list2.add(i);

}

### **Flatmap :**

**To perform operation on stream of stream we use flatmap**

**When there is situation like list of list or list of multidimensional array to perform stream operation on that we use flatmap**

**Ex.**

ArrayList< Integer> list1 = **new** ArrayList<>();

list2.add(12);

list2.add(2);

List<Integer> list2 = **new** ArrayList<Integer>();

list3.add(9);

list3.add(91);

List<List<Integer>> lists = **new** ArrayList<>();

lists.add(list1);

lists.add(list2);

Stream<List<Integer>> liStream = lists.stream();

Stream<Integer> flatMap = liStream.flatMap(e->e.stream());

List<Integer> collect =

flatMap.filter(e-> e<10).collect(Collectors.*toList*());

System.***out***.println(collect);

Q.

# Date & Time Api :

**in 1.8version Joda-Time API is introduced**

**LocalDate date = LocalDate.*now*();**

**LocalTime time = LocalTime.*now*();**

**LocalDateTime dTime = LocalDateTime.*now*();**

**System.*out*.println(time.getHour()+":"+time.getMinute());**

**System.*out*.println(date);**

**System.*out*.println(dTime);**

## **Q.set manual time**

LocalDateTime dTime2 =

LocalDateTime.*of*(1997, 10, 21, 03, 15, 00);

//yyyy mm dd hh mm ss

## **Q.Programe to check time between two dates**

**LocalDate dobDate = LocalDate.*of*(1997, 10, 21);**

**Period period = Period.*between*(dobDate, date);**

**System.*out*.println(period.getDays()+" / "+period.getMonths()+" / "+period.getYears());**

## **Q.programe to check leap year**

**Scanner sc = new Scanner(System.*in*);**

**int int1 = sc.nextInt();**

**Year year = Year.*of*(int1);**

**if (year.isLeap()) {**

**System.*out*.println("year is leap ");**

**}else {**

**System.*out*.println("year is not leap");**

**}**

# Date formater class :

**Ex.**

LocalDateTime dTime = LocalDateTime.*now*();

DateTimeFormatter dateTimeFormatter =

DateTimeFormatter.*ofPattern*("dd-MMMM-yyyy;HH:mm;EEEE");

String format = dTime.format(dateTimeFormatter);

## For zones :

ZonedDateTime zDateTime = ZonedDateTime.*of*(dTime2,ZoneId.*of*("Asia/Aqtau"));

Zone id:

America/Panama

America/Denver

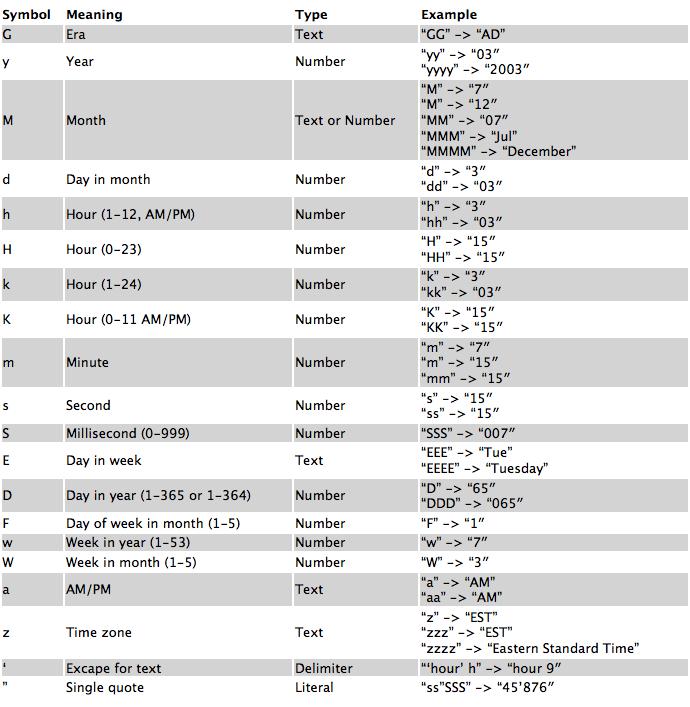
America/New\_York

America/Toronto

America/Los\_Angeles

Asia/Calcutta

Europe/Istanbul



## Q.Convert date to another time zone ?

//create custom date and time

LocalDateTime dateTime1 = LocalDateTime.*of*(1997, Month.***OCTOBER***, 21, 3, 15);

//set timezone of given date

ZonedDateTime dateTime2 = dateTime1.atZone(ZoneId.*systemDefault*());

//set custom time zone by zone id

ZonedDateTime dateTime3 = dateTime2.withZoneSameInstant(zId);

//set date and time format

DateTimeFormatter formatter =

DateTimeFormatter.*ofPattern*("dd-MM-yyyy;HH:mm;E;z");

System.***out***.println(formatter.format(dateTime3));

## Q. convert string to date :

String dateString1 = "10-02-2010;11:20;Europe/London";

String dateString2 = "10-02-2010;11:20";

LocalDateTime dateTime1 = LocalDateTime.*parse*(dateString2, DateTimeFormatter.*ofPattern*("dd-MM-yyy;HH:mm"));

//or

ZonedDateTime dateTime2 = ZonedDateTime.*parse*(dateString1, DateTimeFormatter.*ofPattern*("dd-MM-yyy;HH:mm;z"));

System.***out***.println(dateTime1);

# Java 11 features

**New Utility Methods in String Class :**

**public String repeat(int count) :**

**public boolean isBlank()**

**Local-Variable Syntax for Lambda Parameters:**

**we can declare variables with out providing data type explicitly and just by using "var”**

**EX: var str = "Durga";**

**var i = 10;**

**Flight Recorder:**

**Java Flight Recorder is a profiling tool used to gather diagnostics and profiling data from a running Java application**

**it is available for only for Paying Users, not for normal Users.**

# JDBC:

Dependancy : - com.oracle.database.jdbc

## Demo Program :

Connection connection ;

if (dbUrl.toLowerCase().contains(“pg”)) {

PGDataSource dataSource = getPgDatasource();

dataSource.setUser(userName);

dataSource.setPassword(password);

dataSource.setURL(dbUrl);

connection = dataSource.getConnection();

} else if (dbUrl.toLowerCase().contains(“oracle”)) {

OracleDataSource ods = getOracleDataSource();

ods.setUser(userName);

ods.setPassword(password);

ods.setURL(dbUrl);

connection = ods.getConnection();

}

Statement stmt = connection != null ? connection.createStatement() : null) {

//if we want output from query

ResultSet rs = stmt.executeQuery("select \* from student name");

rs.next();

dbId = rs.getLong(1);

//if we dont want output from query

stmt.execute("DROP SCHEMA IF EXISTS " + pgSchema + " CASCADE");

## Statement vs prepare statement

**statement**: Each time the sql query is running,this sql statement is sent to the DBMS where it is compiled. So, it increases the server loads and decreases the performance.

connection con=null;

String sql="select \* from employee where id=5";

Statement st=conn.createStatement();

**PreparedStatement**: Unlike Statement PreparedStatement is given a sql query as a parameter when it is created.

connection con=null;

String sql="select \* from employee where id=?";

PreparedStatement ps=conn.prepareStatement(sql);

This sql statement is sent to Database where it is compiled. So,in preparedStatement compiled happens only once but in statement compiled happens each time Statement is called.

## RowMapper vs result set

**RowMapper:** To process one record of ResultSet at a time.

while [RowMapper](http://static.springsource.org/spring/docs/current/javadoc-api/org/springframework/jdbc/core/RowMapper.html) is feeded with row at a time.

**ResultSetExtractor:** To process multiple records of ResultSet at a time.

Basic difference is with ResultsetExtractor you will need to iterate through the result set yourself, say in while loop

# Spring

## **Features :**

**light weight**

**loosely coupled**

**reduce boiler plate code**

**easily testable because of its own container**

## **Core Module :**

**Spring Ioc : (inversion of control)**

**It means rather then we creating object spring will create object for us**

**Spring beans :**

**in spring java objects are called beans, the objects which are store in container are called beans**

**IOC Container :**

**It is responsible to create object**

**The IoC container gets informations from the XML file and annotations, and it works according to that.**

## DI :

Dependency injection generally means **passing a dependent object as a parameter to a method, rather than having the method create the dependent object**.

So that the method does not have a direct dependency on a particular implementation.

Any implementation that meets the requirements can be passed as a parameter.

EX.

Consider a class Employee

class Employee {

private int id;

private String name;

private Address address;

Employee() {

id = 10;

name="name";

address = new Address();

}

}

and consider class Address

class Address {

private String street;

private String city;

Address() {

street="test";

city="test1";

}

}

In the above code the address class values will be set only when the Employee class is instantiated, which is dependency of Address class on Employee class.

And spring solves this problem using Dependency Injection concept by providing two ways to inject this dependency.

**1.Setter injection :**

Setter method in Employee class which takes a reference of Address class

public void setAddress(Address addr) {

this.address = addr;

}

**2.Constructor injection**

Constructor in Employee class which accepts Address

Employee(Address addr) {

this.address = addr;

}

In this way the Address class values can be set independently using either setter/constructor injection.

## **There are two types of IoC containers. They are :**

**BeanFactory**

**ApplicationContext**

|  |  |
| --- | --- |
| **Bean Factory(I)** | **Application Context(I)** |
| **It is basic IOC container**  **The XmlBeanFactory is the implementation class for the BeanFactory interface.**  **Resource resource = new ClassPathResource(“applicationContext.xml”)** | **It is build on top of bean factory**  **It have some extra functionality then bean factory :**  **Simple intregreation with other spring module**  **Application layer specific context like webApplicationContext for web application**  **The ClassPathXmlApplicationContext class is the implementation class of ApplicationContext interface.**  **ApplicationContext context =**  **new ClassPathXmlApplicationContext("applicationContext.xml");** |

## **Constructor vs setter injection**

|  |  |
| --- | --- |
| **Constructor injection** | **Setter injection** |
| **Partial Injection not possible (if there are 3 parameters in constructor and you only want to pass one this is not passible)**  **Does not override the setter property**  **Creates new instance if any modification occurs**  **Better for too many properties** | **Partial Injection possible**  **Overrides the constructor property if both are defined (Setter injection overrides the constructor injection. If we use both constructor and setter injection, IOC container will use the setter injection.)**  **Doesn't create new instance if you change the property value**  **(We can easily change the value by setter injection. It doesn't create a new bean instance always like constructor. So setter injection is flexible than constructor injection.)**  **Better for few properties.** |

## **AutoWiring :**

**it enables the programmer to inject the bean automatically.**

**We don't need to write explicit injection logic for that** .

**<bean id="emp" class="com.javatpoint.Employee" autowire="byName" />**

## **Autowireing Modes :**

|  |  |  |
| --- | --- | --- |
| 1) | byName | injects the bean based on the property name. It uses setter method. |
| 2) | byType | injects the bean based on the property type. It uses setter method.(by datatype) |
| 3) | BYconstructor | It injects the bean using constructor |

## **Bean scope :**

### **1)Singleton Scope:**

**it is default scope**

**in this bean will create only one object and when everytime we call getBean()**

**it will pass that same object everytime we call**

**it initialize when container start**

### **3)Proto-Type Scope :**

**we will get brand-new bean everytime we call getBean()**

**it initialize when we call getBean()**

**it is also gets initialized when it is used as a dependency inside a singleton scope bean**

#### Scoped Proxy :

**beanA has the singleton scope and beanB has prototype scope.**

**beanA has the beanB dependency.**

**it provides the random behavior as beanA will have a single instance for the container, but beanB will not have the same.**

**Whenever the beanB is requested from the container a new instance will be created.**

**To solve this we use scoped proxy,**

**Spring framework provides the @Scope annotation for specifying the scope. proxyMode is the annotation attribute to specify the proxy type**.

**Ex. @Scope(value="prototype", proxyMode=ScopedProxyMode.TARGET\_CLASS)**

### **3)Request Scope :**

**it will create new bean everytime for every new http request**

**@Request Scope**

### **4)Session Scope :**

**it will create new bean for every new Session**

**session scope is very similar to HttpSession .**

**so the Beans which instantiated based on session scope, it lives throughout the HTTP session.**

**Similar to request scope, it is applicable only for web aware spring application contexts.**

**@Session Scope**

### **5)Global Session Scope :**

**your application works in a Portlet container it is built of some amount of portlets.**

**Each portlet has its own session, but if you want to store variables global for all portlets in your application**

**then you should store them in globalSession.**

**This scope is same as session scope in Servlet based applications.**

## **Bean Life Cycle :**

**IoC container instantiates by the bean definition in the XML file.**

**Spring perform dependency injection**

**Init() method called (it will do all the specified initialization stuff like connecting to DB or services)**

**Bean ready to use**

**In the end it will call Destroy() method (which perform cleanup operations)**

### **There are mainly two ways to configure bean Life cycle :**

**1) Annotation based :**

**@postConstruct : annotated method will be invoked after the bean has been constructed and just before it’s instance is returned to requesting object.**

**@preDestroy : annotated method is invoked just before the bean is about be destroyed inside bean container.**

**Xml approach :**

**We have to define these methods in Xml file**

**Life cycle method are also called as callback methods as those methods are automatically called by framework.**

**RegisterShutDownHook() :**

**It will execute once when main method ended , so once all the code gets executed it will called and close your ioc container, so it wont give us any exception irrespective of the line number on which we are calling it .**

## **Spring Annotations :**

**@autowire : we can use this to set automatic object reference in java file**

**First it will try to resolve this by type**

**If it fails then it goes with byName**

**@Required: It applies to the bean setter method.**

**It indicates that the annotated bean must be populated at configuration time with the required property.**

**@Configuration: It is a class-level annotation.**

**Spring Containers use this annotation as a source of bean definitions.**

**@ComponentScan: It is used when we want to scan a package for beans.**

**It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.**

**@Bean: It is a method-level annotation.**

**It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.**

**Ex.**

**@configuration**

**Public class Config{**

**Public Student studying(){**

**Student S1 = new Student()**

**return student;**

**}**

**}**

**@Component: It is a class-level annotation.**

**It is used to mark a Java class as a bean.**

**@Service: It is also used at class level.**

**It tells the Spring that class contains the business logic.**

**@Repository: It is a class-level annotation.**

**The repository is a DAOs (Data Access Object) that access the database directly.**

**The repository does all the operations related to the database.**

## **Spring Event :**

**Events are designed for simple communication among Spring beans within the same application context.**

**(example: sending an email on some task completion). Also, events come in handy for test-driven development.**

**Spring Framework internally use a lot of these events to communicate various states**

### **1. Spring Events vs Method Call**

**When communicating with other modules, we have the following options while working on a Spring application.**

**Using traditional method call.**

**Using Spring framework events system**

**Method call is a traditional or most common way while communicating.**

**In most cases we may need method call where we need response from the other components before we can proceed to the next step**

**(e.g Verifying credit card information during the checkout.**

**Events on the other hand in a way to notify all parties about a specific steps happens in the process so that all interested parties can take actions.**

**Ex.**

**Sending out emails to the customer on order placement.**

**Sending out notification when order is shipped.**

### **2. Custom Spring Events**

**2.2 Custom Events Listener**

**2.3 Publishing Events**

### **3. Async Events**

**these events work in Async mode, which means publisher thread will block until all the listeners (who are listing to this event) have finished processing this event.**

### **4. Filtering**

**We can also use condition attribute on our listener to filter certain events**

#### **5. Transaction bound events**

**Spring provides a way to bound events to a certain phase of a transaction (e.g. publish an event when a transaction is complete**

## Spring - Transaction Management :

Transaction management is an important part of enterprise application which connected to database to ensure data consistency.

### The concept of transactions based on ACID properties :

**Atomicity − like A transaction should be treated as a single unit of , which means either the operations is successful or unsuccessful.**

**Consistency − represents the consistency of integrity of the database,**

**Which consist unique primary keys in tables**

**Isolation − There may be many transaction processing with the same data so Each transaction should be isolated from others to prevent data corruption.**

**Durability − Once a transaction has completed, the results of this transaction have to be made permanent like it cannot be erased from the database due to system failure or anything.**

**Spring's transaction help us by adding transaction capabilities to POJOs.**

***PlatformTransactionManager is a interface which provide us methods like :***

**getTransaction(TransactionDefinition definition)**

**This method returns a currently active transaction or creates a new one,**

**void commit(TransactionStatus status)**

**This method commits the given transaction, with regard to its status.**

**void rollback(TransactionStatus status)**

**This method performs a rollback of the given transaction.**

# Spring MVC :

**It is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern.**

**A Spring MVC works with the help of DispatcherServlet.,it is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.**

**Model -it model contains the data of the application. A data can be a single object or a collection of objects.**

**Controller -it contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.**

**View - A view represents the provided information in a particular format. Like Jsp thymeleaf page**

**Front Controller - In Spring Web MVC, the DispatcherServlet class works as the front controller and It is responsible to manage the flow of the Spring MVC application.**

**Steps to create spring Mvc Application :**

**Configure dispatcher servlet in web.xml**

**Create spring configuration file**

**Configure view resolver**

**Create controller**

**Create view to show page**

**Model class : it contains addAttribute(string key,object obj) method to send data from controller to view**

**Redirecting in MVC :**

**httpServletResponce : by using servlet**

**RedirctPrefix : adding Redirect before/……request**

**RedirectClass : redirectobj.setUrl(“ ”); return redirectobj;**

**What kind of configuration we can do in web.xml ?**

**Configure servlet, url mapping, filters, security, error handling, welcome file list**

# Spring JDBC :

**Spring JdbcTemplate eliminates all the problems of JDBC API**

**We need to write a lot of code before and after executing the query,**

**such as creating connection, statement, closing resultset, connection etc.**

**Repetition of code ,it is a time consuming task**.

**JdbcTemplate class**

**It is the central class in the Spring JDBC support classes.**

**We can perform all the database operations by the help of JdbcTemplate class method**

**Update() : it used to perform insert update delete operation**

**Execute() : it used for select \* from type of queries**

**Batchupdate() : to insert array or collection**

**Query() : if we want to retrive all records from database**

**Result set : A ResultSet object points to the current row in the result set.**

**Ex.**

**Connection con=DriverManager.getConnection("localhost:1521","system","oracle");**

**Statement stmt=con.createStatement(ResultSet…..,ResultSet.xyz);**

**ResultSet rs=stmt.executeQuery("select \* from emp765");**

|  |  |
| --- | --- |
| **RowMapper** | **ResultSetExtractor** |
| **To process one record of ResultSet at a time.** | **To process multiple records of ResultSet at a time.** |

# @Transactional :

**If there is any exception in programme ,this annotation help us to rollback changes before exception**

**Ex.**

**trying to book a ticket all personal info is stored but there is issue in payment, so in this case the data is not commited till the whole process ends smoothly, if there is any exception in this case ex. payment not successed then whole data will be rollback.**

**We can give method and class @Transactional annotation**

**And @EnableTransactionManagement on main class**

**By default it only works for runtime exceptions and errors, but if we want to exclude specific exception :**

**@Transactional(noRollBackFor= Exception.class(mention specific exception here))**

**We can specify Isolation level :**

defines how the changes made by one transaction affect other simultaneous concurrent transactions,

and also when that changed data becomes available to other transactions.

### (Dirty REad)

READ\_UNCOMMITTED : means that a transaction may read data that is still uncommitted by other transactions.

READ\_COMMITTED : means that a transaction can't read data that is not yet committed by other transactions.

@Transactional(isolation=Isolation.READ\_COMMITTED)

public void someTransactionalMethod(Object obj) {

}

**We can specify Propagation** :

is the ability to decide how the business methods should be participate or not in transactions.

REQUIRED : behavior means that the same transaction will be used if there is an already opened transaction in the current bean method.

REQUIRES\_NEW : behavior means that a new physical transaction will always be created by the container.

# Spring Data Jpa :

**The Java Persistence API (JPA) is the specification of Java that is used to persist data between Java object and relational database.**

**JPA acts as a bridge between object-oriented domain models and relational database systems.**

**As JPA is just a specification, it doesn't perform any operation by itself. It requires an implementation.**

**Therefore, ORM tools like Hibernate, TopLink, and iBatis implements JPA specifications for data persistence.**

## **ORM :**

**The object-relational mapping is a mechanism which is used to develop and maintain a relationship between an object and the relational database, by converting objects into columns.**

**The advantages of JPA are given below :**

**The burden of interacting with the database reduces significantly by using JPA.**

**Reduce programming**

**We can merge the applications used with other JPA providers**

**@Embeddable :**

**When we use this to embed and object inside another table**

**Ex School Pojo have inside Student pojo object**

**JPQL :**

**JPQL is the Java Persistence query language defined in JPA specification. It is used to construct the queries.**

## **What are the different directions of entity mapping?**

**unidirectional mapping : only one entity can be mapped to another entity**

**bidirectional mapping : both entity can be mapped or referred to another entity.**

**One-to-one mapping: a instance of one entity is associated with an instance of another entity.**

**Ex. one man can have one nose**

**One-To-Many mapping:  the instance of one entity can be mapped with any number of instances of another entity with the help of collections.**

**Ex.  A one cart can have multiple items in it**

**Many-to-one mapping The Many-To-One mapping represents a single-valued association where a collection of entities can be associated with the one entity**

**Ex. Many students can have one teacher**

**Many-to-many mapping  any number of collection of entities can be associated with a collection of other entities.**

**Ex. many teacher can teach many subjects**

## [**Cascade**](https://www.tutorialandexample.com/cascade-in-hibernate) **:**

**it is used to work with related entities**

**Ex. one question can have many answers so when we try to save question in DB all the answers also have gets automatically saved we don’t have to do it manually**

**here cascade comes handy**

**So to work with related entities we use cascade.**

**CascadeType.ALL**

**It is used to propagate all operations (save, persist, merge, refresh, etc.) from a superclass to a subclass.**

**CascadeType.DETACH**

**When we use CascadeType.DETACH operation; the subclass object will get removed from the persistent state.**

## **orphan removal in mappings:**

**in one-to-one or one-to-many mapping is removed, then remove operation can be cascaded to the target entity. Such target entities are known as orphans, and the orphanRemoval attribute can be used to specify that orphaned entities should be removed.**

|  |  |
| --- | --- |
| **Lazy Loading** | **Eager Loading** |
| **when we enable lazy loading, if we pull up a entity Data, the data of associated entities won't be initialized and loaded into a memory until we make an explicit call to it.**  **Smaller load time**  **Less memory consumption** | **In eager loading strategy, if we load the Entitydata, it will also load up all entities associated with it and will store it in a memory.**  **No delayed initialization so it will not cause performance impacts** |

## **Object states/persistence life cycle of an object :**

**Transient - when object is declared by using the new keyword. When an object remains in the transient state, it doesn't contain any (primary key) in the database.**

**Persistence - In this state, an object is saved to a database or retrieved from the database. When an object remains in the persistence state, It contains a row of the database and consists of an primary key value value.**

**Detached - The object enters into a detached state when the hibernate session is closed. The changes made to the detached objects are not saved to the database.**

## **Types of identifier generation :**

**We use this with @GeneratedValue annotation:**

**Automatic Id generation - In this case, If any value is not specified explicitly, the generation type defaults to auto.**

**Id generation using a table - The identifiers can also be generated using a database table.**

**Id generation using a database sequence - Databases support an internal mechanism for id generation called sequences. To customize the database sequence name, we can use the JPA @SequenceGenerator annotation.**

**Id generation using a database identity - In this approach, whenever a row is inserted into the table, a unique identifier is assigned to the identity column that can be used to generate the identifiers for the objects.**

**Configure Jpa in Spring Boot : add data jpa dependency :**

**1)create user class : give @entity annotation to class and @id(auto generate to id which is primary key)id datatype should be int**

**2)configure datasource in application.properties file.**

**3)create userRepository interface: extend it with CRUDRepository or jpaRepository**

**4)create bean/obj of userRepository(use that obj to perform crud operation)**

## **Jpa Methods:**

**It provide us inbuilt methods :**

**FindAll()**

**FindById()**

**We can also create custom finder method :**

**Ex. select \* from table\_name where name = “…” :-**

**Public list<user> findByName(String name);**

**Public list<user> findNameStratingWith(String prefix);**

**Public list<user> findNameEndingWith(String suffix);**

**Public list<user> findAgeLessThen(int age);**

**Also we use : Public list<user> findByNameAndCity(String name, String name);**

**Can use keywords like and ,or**

**Public list<user> findByAgeIn(Collection<integer> ages);**

**Writing sql queries in Jpa (jpql):**

**@Query(“select u from User u WHERE u.name= :a and u.city= :b”)**

**Public list<user>getUser(@param(“a”)String name, @param(“b”)String name);**

**Writing native query(sql ):**

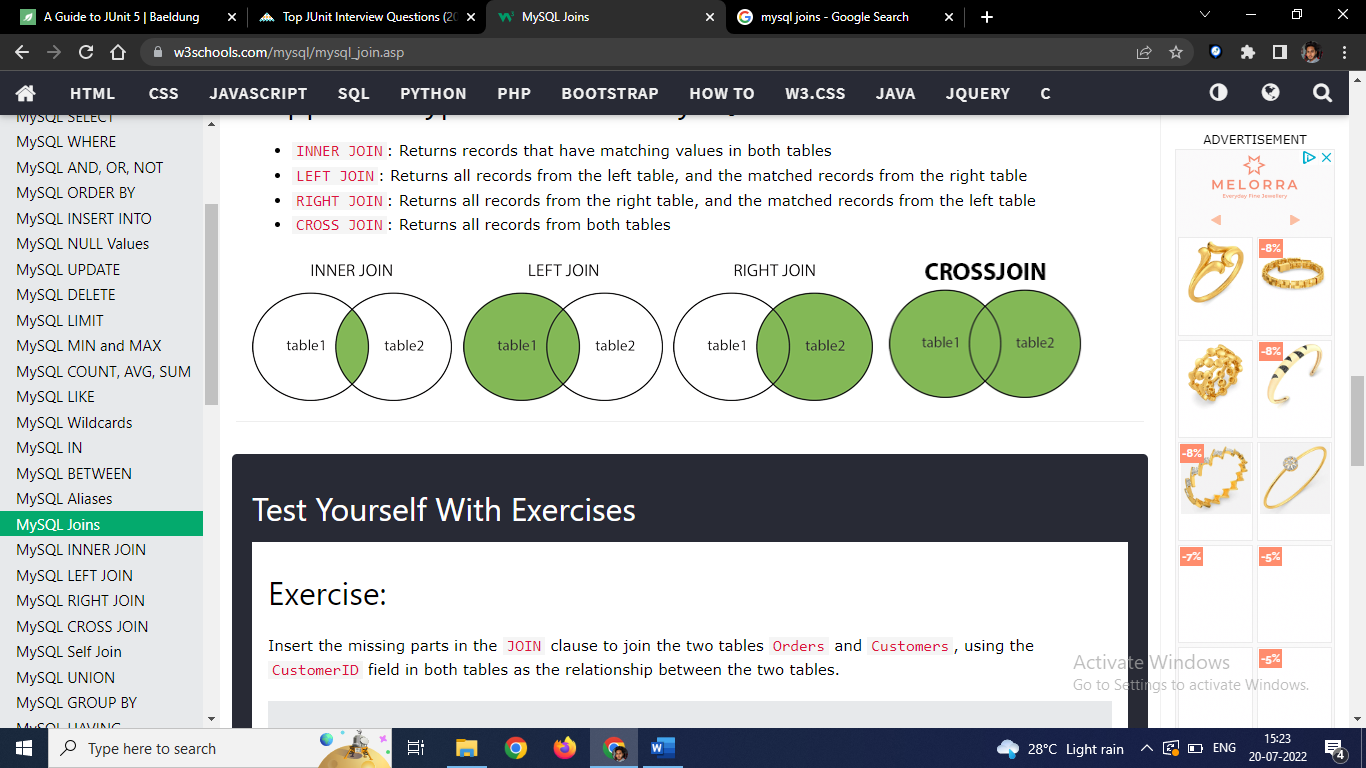
**@Query(“select \* from user”,nativeQuery = true)**

**Public list<user> getAll();**

## **Joins :**

**Join query is used to combine rows from two or more tables ,based on related columns between them**

**There also a One condition atleast one column should be common in both the table**



**INNER JOIN: Returns matching values in both tables**

**Ex.**

**SELECT** table1.column(1), table1.column(2), table2.column(1), table2.column(2).. so on **FROM** table1 **INNER JOIN** table2 **ON** table1.MatchColumnName = table2.MatchColumnName;

**LEFT JOIN: Returns all records from the left table, and the matched records from the right table**

**Ex.**

**SELECT** table1.column(1), table1.column(2), table2.column(1), table2.column(2).. so on **FROM** table1 **LEFT JOIN** table2 **ON** table1.MatchColumnName = table2.MatchColumnName;

**RIGHT JOIN: Returns all records from the right table, and the matched records from the left table**

**Ex.**

**SELECT** table1.column(1), table1.column(2), table2.column(1), table2.column(2).. so on **FROM** table1 **RIGHT JOIN** table2 **ON** table1.MatchColumnName = table2.MatchColumnName;

**CROSS JOIN: Returns all records from both tables**

**Ex.**

**SELECT** table1.column(1), table1.column(2), table2.column(1), table2.column(2).. so on **FROM** table1 **LEFT JOIN** table2

**ON** table1.MatchColumnName = table2.MatchColumnName **UNION SELECT** table1.column(1), table1.column(2), table2.column(1), table2.column(2).. so on **FROM** table1 **RIGHT JOIN** table2 **ON** table1.MatchColumnName = table2.MatchColumnName;

# Spring Boot

|  |  |
| --- | --- |
| **Spring** | **Spring Boot** |
| It is most widely use Java EE  framework  For building or developing applications.  It aims to simplify Java EE development that makes developers more productive.  primary is **dependency injection**.  It helps to make things simpler by allowing us to develop **loosely coupled** applications.  The developer has writes a lot of code (**boilerplate code**) to do the minimal task.  To test the Spring project, we need to set up the sever explicitly.  Developers manually define dependencies for the Spring project in **pom.xml**. | For developing REST APIs Spring Boot framework is widely used.  It aims to shorten the code length and provide the easiest way to develop **Web Applications**.  The primary feature is **Autoconfiguration**. It automatically configures the classes based on the requirement.  It helps to create a **stand-alone** application with less configuration.  It **reduces** boilerplate code.  It offers **embedded server** such as **Jetty** and **Tomcat**, etc.  Spring Boot comes with the concept of **starter** in pom.xml file that internally takes care dependencies  based on Requirement. |

## **Application vs yml**

|  |  |
| --- | --- |
| **Application.properties** | **Yml.properties** |
| .properties stores data in sequential format  It supports only key-value pairs (basically string values)  .properties is specifically used by Java | .yml stores data in hierarchical format  It supports key-value pair, as well as map, list & scalar type values  .yml can be used by other languages (eg Java, Python, etc). |

**Web services** are client and server applications that communicate over the Web (WWW) using HyperText Transfer Protocol (HTTP).

The server can process the **request** and sends the **response** to client

We can exchange data using XML and Json Format.

Read values from properties file :

@value(value=”${app.name}”)

Private string name;

change port in boot in properties file:

server.port=8081

change default server in boot :

update pom.xml,

add the dependency for spring-boot-starter-jetty.

Also, we need to exclude default added spring-boot-starter-tomcat dependency.

## **Features :**

**We can create standalone application using boot and store it in java jar files (we don’t need war file to deploy)embedded sever can directly deploy jar files**

**Boot automatically configure spring and third party library whenever possible**

**No requirement of xml configuration**

**Spring boot uses starter pom dependencies which contains jars with prebuild configuration**

## Spring boot starter Dependancy :

|  |  |
| --- | --- |
| **spring-boot-starter-data-jpa** | Starter for using Spring Data JPA with Hibernate |
| **spring-boot-starter-web** | Starter for building web  including RESTful  applications using Spring MVC.  Uses Tomcat as the default embedded container. |
| **spring-boot-starter-tomcat** | Starter for using Tomcat as the embedded servlet container. Default servlet container starter used by spring-boot-starter-web |
| spring-boot-devtools | It provide features such as live reload  DevTools pick up the changes and restart the application.  It saves time |
| **spring-boot-starter-logging** | Starter for logging using Logback. Default logging starter |

## Spring profile :

There are some application properties which vary per environment,so to handle that with we use spring profile

Ex.

There are enviroments like dev,qa,prod so for every stage there will be a different set of configuration like different database or different port so to handel this we create seprate properties file

Application.properties

Application-dev.properties

Application-Prod.properties

We have to mention in our defaut Application.properties which profile we want to activate

Spring.profile.Active = dev

@profile :

If we want to create environment specific bean then we use this annotation

Ex.

@profile(value = {“dev”}

@Bean(name=”myprop”)

Public string creatatebean(){

Return “bean123”;

}

**Two types of web services :**

## **Soap vs rest**

|  |  |
| --- | --- |
| **Soap** | **Rest** |
| Simple Object Access Protocol.  the data exchange format is always XML.  are typical to implement.  SOAP is a protocol. | REpresentational State Transfer.  There is no strict data exchange format. We can use JSON, XML, etc.  are easier to implement than SOAP.  REST is an architectural approach. |

## **REST FULL web Services :**

**RESTful** Web Services are basically REST Architecture based Web Services.

they are light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications.

**Api :** it is set of rules that allows programe’s talk to each other

**RestAPi :**

**it is based on client server architecture**

**it is stateless means server can not store client data**

**we can use it with any server**

**it is cacheable which reduces request** 🡨🡪 **response which increases speed**

**HTTP methods :**

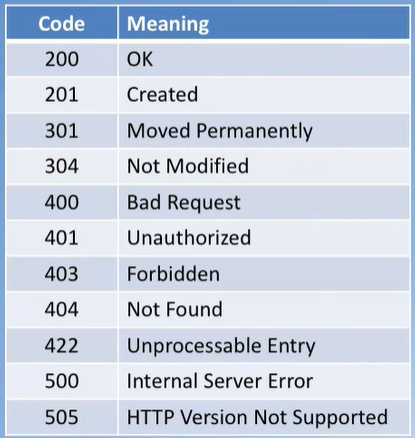
**Get : it reads the resource**

**Put : it updates and existing resource**

**Post : it creates a new resource**

**Delete : it Deletes the resource**

**Http Status codes :**

****

## **Json :**

**Java script object notation**

**It is sort of java script object**

**All the request and response from client to server are in Json format**

**Spring boot automatically converts java objects into jSon by using Jackson dependency**

**Response Entity :**

it **represents the whole HTTP response: which mostly includes status code ,messages and response body**

If we want to use it, we have to return it from the endpoint, and Spring takes care of the rest.

**Ex.**

ResponseEntity<String> **hello**() {

**return** **new** **ResponseEntity**<>("Hello World!", HttpStatus.OK);

}

## Json vs xml :

|  |  |
| --- | --- |
| **Json** | **Xml** |
| * Simple syntax, * Easy to use with JavaScript * JSON is both more compact and (in my view) more readable also * Also in transmission it can be "faster" simply because less data is transferred. | * The XML (extensible Markup Language) is kind of a standard broadcasting language, which can be used by any programming language, and supported both server and client side, so it is quite flexible solution * In my guess The XML is best for the bigger websites, for example shopping sites or something like this. * The XML can be more secure and clear. |

## **Spring Actuator :**

**It is kind monitoring application for rest services**

We can monitor health, CPU usage, HTTP hits these all are automatically applied to boot application

spring-boot-starter-actuator add dependency

hit request from postman ex. http://localhost:8080/**Beans**

Actuators provide below pre-defined endpoints to monitor our application -

Health

Info

Beans

Mappings

Configprops

Httptrace

Heapdump

Threaddump

Shutdown

## **Annotations :**

**@SpringBootApplication :**

**@EnableAutoConfiguration + @ComponentScan +** **@Configuration**.

**@RequestMapping:** It is used to map the **web requests**.

**@GetMapping:** It maps the **HTTP GET** requests on the specific handler method.

**@PostMapping:** It maps the **HTTP POST**requests on the specific handler method

**@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method.

**@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method.

**@PatchMapping:** It maps the **HTTP PATCH**requests on the specific handler method. **is used when you want to apply a partial update to the resource**

**@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter.

**@ResponseBody:** It binds the method return value to the response body. It tells the Framework return an object into JSON and XML format.

**@PathVariable:** It is used to extract the values from the URI.

**Ex.**

@GetMapping("/api/employees/{id}")

@ResponseBody

**public** String **getEmployeesById**(@PathVariable String id) {

**return** "ID: " + id;

}

**@RequestParam:** It is used to extract the query parameters form the URL.

**Ex.**

@GetMapping("/api/foos")

@ResponseBody

**public** String **getFoos**(@RequestParam String id) {

**return** "ID: " + id;

}

**@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a **method parameter**. The optional elements of the annotation are **name, required, value, defaultValue.**For each detail in the header, we should specify separate annotations. We can use it multiple time in a method

**@RestController:** It can be considered as a combination of **@Controller** and **@ResponseBody**annotations**.** The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.

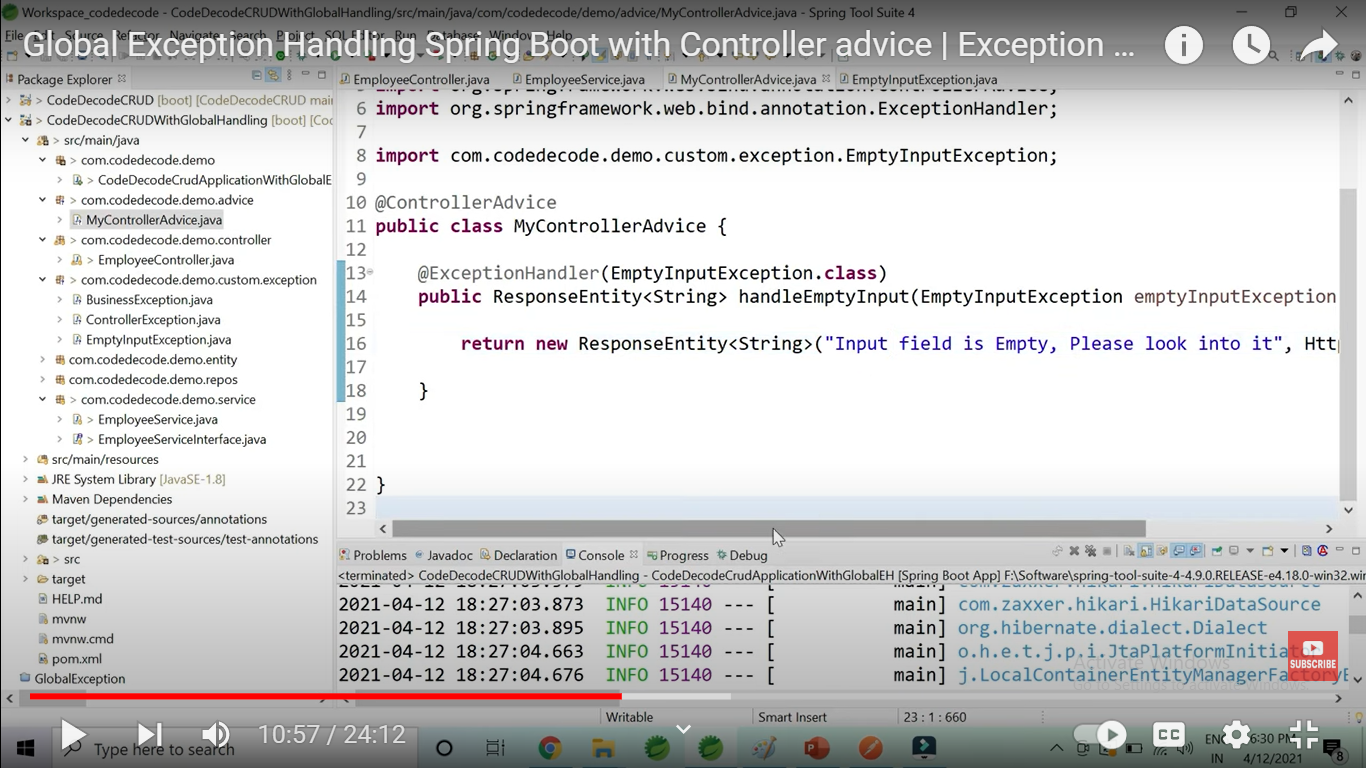
**@ModelAttribute:**

it refers to a property of the Model object (the M in MVC ;) so let's say we have a form with a form backing object that is called "Person" Then you can have Spring MVC supply this object to a Controller method by using the @ModelAttribute annotation.

## **Global Exception Handling in Spring boot :**

**We have to give @ControllerAdvice to class**

**And in this class we have to specify our custom exception which will get’s handle throughout the programme.**

****

## RestTemplate :

is is used to called http methods in synchronous way

Ex:

// Creating an instance of RestTemplate class

RestTemplate rest = new RestTemplate();

// Method

public UserData getUserData()

{

return rest.getForObject(

"http://localhost:8080/RestApi/getData",

UserData.class);

}

// Method

public ResponseEntity<UserData> post(UserData user)

{

return rest.postForEntity(

"http://localhost:8080/RestApi", user,

UserData.class, "");

}

}

### How to send header :

ClientHttpRequestInterceptor:

1.public class HeaderRequestInterceptor implements ClientHttpRequestInterceptor {

private final String headerName;

private final String headerValue;

public HeaderRequestInterceptor(String headerName, String headerValue) {

this.headerName = headerName;

this.headerValue = headerValue;

}

@Override

public ClientHttpResponse intercept(HttpRequest request, byte[] body, ClientHttpRequestExecution execution) throws IOException {

request.getHeaders().set(headerName, headerValue);

return execution.execute(request, body);

}

}

2. List<ClientHttpRequestInterceptor> interceptors = new ArrayList<ClientHttpRequestInterceptor>();

interceptors.add(new HeaderRequestInterceptor("Accept", MediaType.APPLICATION\_JSON\_VALUE));

RestTemplate restTemplate = new RestTemplate();

restTemplate.setInterceptors(interceptors);

## WebClient :

is it also used to call http methods but it can do both sync and async.

For this we need webflux dependency.

Ex.

Step 1:

@Bean  
public WebClient webClient() {  
 return WebClient.builder().baseUrl(addressBaseUrl).build();  
}

Step 2:

AddressResponse addressResponse = webClient.get().uri("/address/" + id).retrieve().bodyToMono(AddressResponse.class).block();  
 employeeResponse.setAddressResponse(addressResponse);  
 return employeeResponse;

# Spring Aop :

[Spring AOP (Aspect Oriented Programming)](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#aop) is kind of [**OOPs**](https://howtodoinjava.com/java/oops/object-oriented-programming/)**,** like it also provides modularity. In OOPs, the key unit is Objects, but **in AOP, the key unit is aspects** or concerns (it is like stand-alone modules in application).

**It consist of :**

**Concerns :** **is like a behavior that we want to have in an application module**

**Ex.** in any eCommerce application, different concerns (or modules) may be inventory management, shipping management, user management etc.

**cross-cutting concern : is a concern that is applicable throughout the application (or more than one module).**

Ex. logging, security and data transfer are the concerns needed in almost every module of an application.

# Log4J :

**It is reliable, fast and flexiable framework it provided by apache foundation**

**Features :**

**It won’t cause any data inconsistency**

**It will not slow down application performance it will optimized to improve application performance**

**It will provide environment to send logging messages to database**

**It will use multiple levels to generate message :**

* **All**
* **Trace**
* **Debug**
* **Info**
* **Warn**
* **Error**
* **Fatal**

## **4 core objects in Log4J :**

**Logger : responsible to take message from java application.**

**Appender : this obj send message to destination ,with this there must be a one destination object to in order to send logging message.**

**Layout : his main objective to form logging message into different style, layout object used in appender just before sending data to destination**

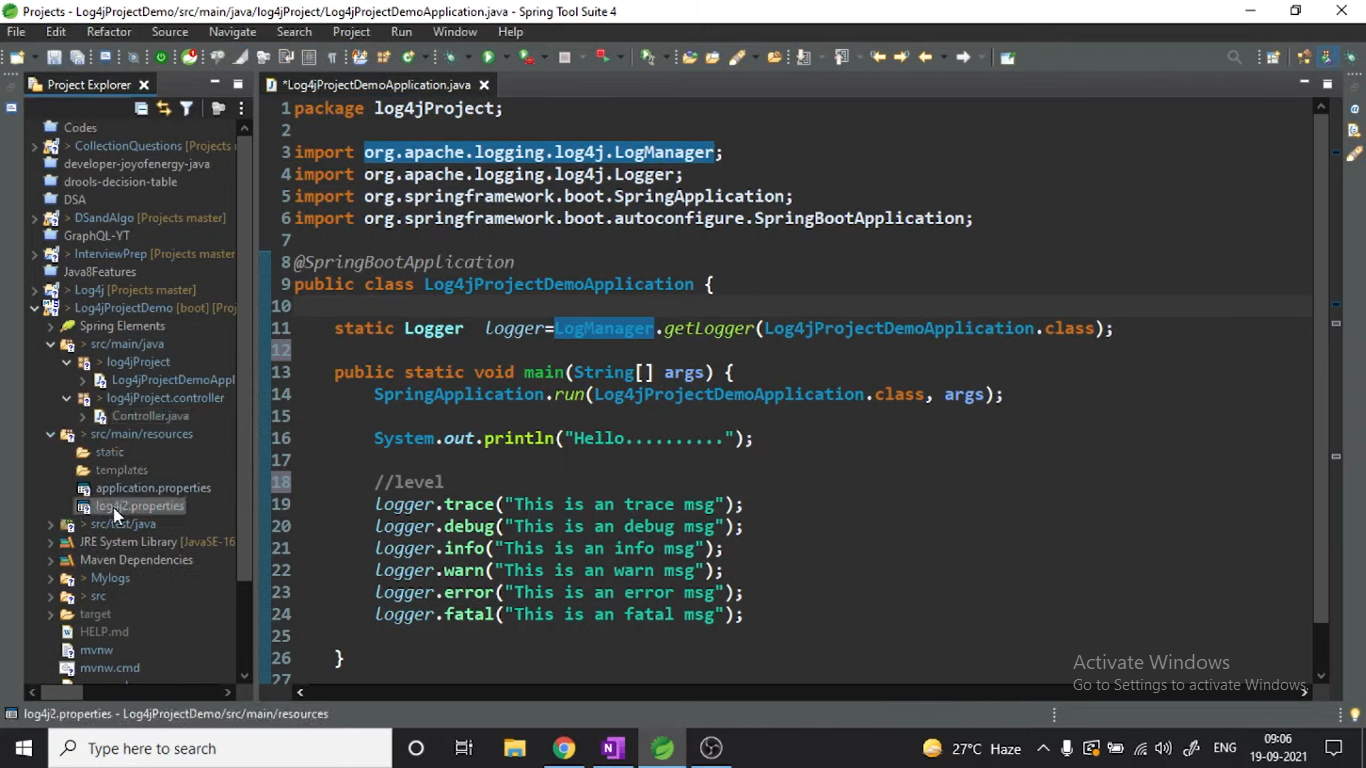
**Destination : where message in finally published**

**Filters :**

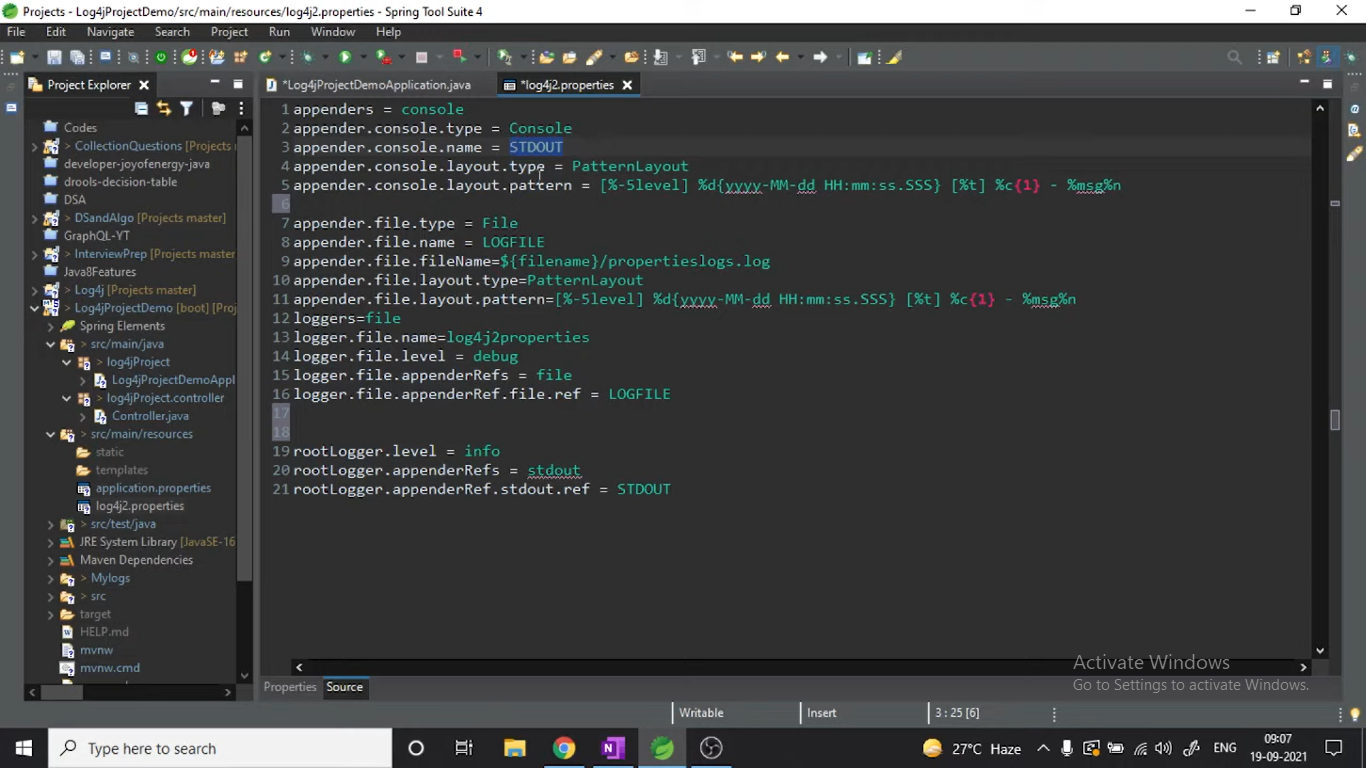
**main intension is to analyze and takes the decision whether the message must be logged or not**

**How to implement Logger :**

**Add log4j dependency**

****

**Properties file**

****

## **Log4j Vulnerablity:**

it allows malicious attackers to execute code remotely on any targeted computer .

**how to fix Log4j Vulnerablity :**

**as we are using java8 we should jumped to version 2.16 to overcome this.**

**Also if we using java 1.7> we have to make changes in log4j.properties file.**

## Log Traceing :

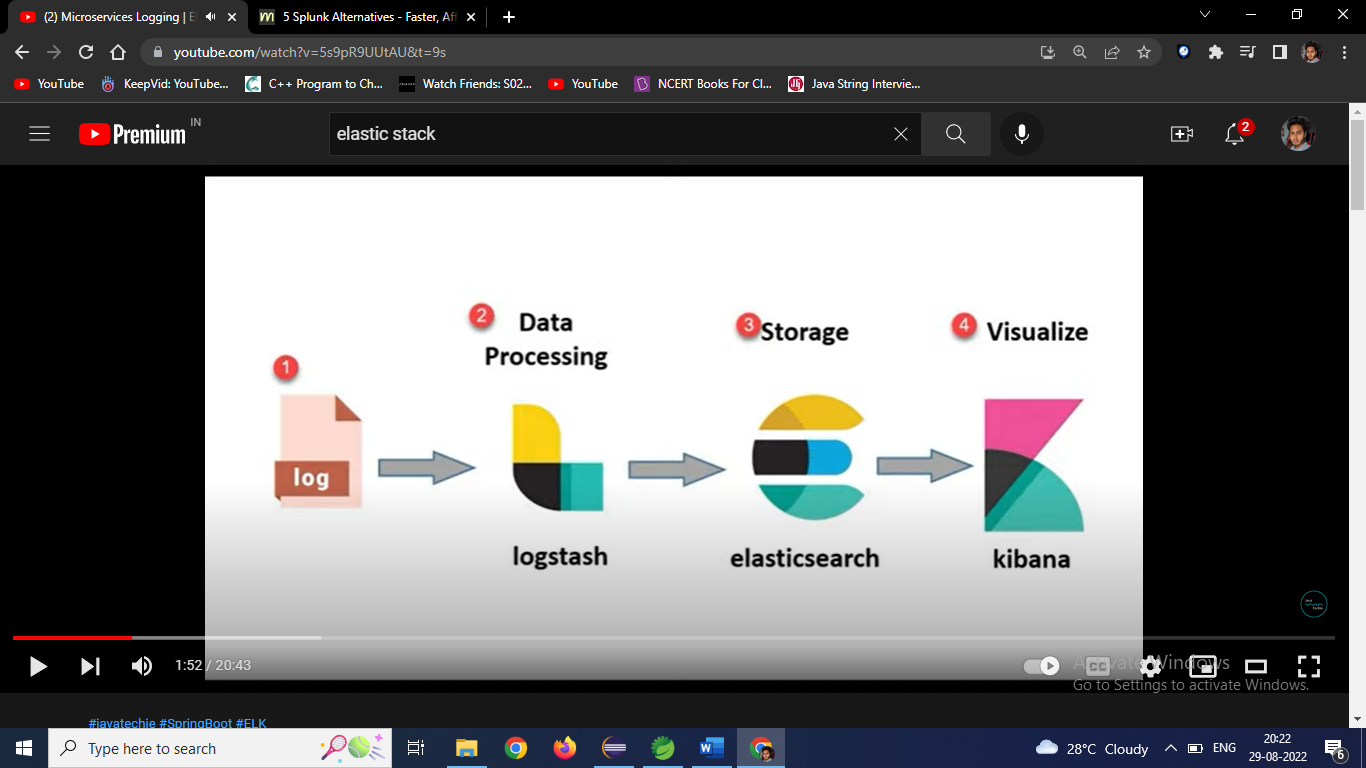
**Elk stack :**

* **Elastic search : it is databse use to store input/logs**
* **Logstash: it is kind of pipeline which import and export data from one data source to another**
* **Kibana : it is kind of a ui tool that helps to monitor log**

**We have access to kibana dashboard ()**

**There are indexes in kibana which shows logs by date and time**

**in which we discovers the logs on dashboard and if we want any specific log we can find it in it.**

****

# Microservices :

|  |  |
| --- | --- |
| **Monolithic architecture** | **Microservices Architecture** |
| * is built as one large system and is usually one code-base * It is not easy to scale based on demand * It has shared database * Large code base makes IDE slow and build time gets increase. * It extremely difficult to change technology or language or framework because everything is tightly coupled and depend on each other | * is built as small independent module based on business functionality * It is easy to scale based on demand. Each project and module has their own database * Each project is independent and small in size. So overall build and development time gets decrease. * Easy to change technology or framework because every module and project is independent |

## **Benefits:**

Self-contained, and independent deployment module.

Independently managed services.

In order to improve performance, the demand service can be deployed on multiple servers.

It is easier to test and has fewer dependencies.

A greater degree of scalability and agility.

Simplicity in debugging & maintenance.

Better communication between developers and business users.

Development teams of a smaller size.

## **Drawbacks:**

Due to the complexity of the architecture, testing and monitoring are more difficult.

Pre-planning is essential.

Complex development.

Expensive compared to monoliths.

Security implications.

Maintaining the network is more difficult.

## Microservices design pattern :

### SAGA Pattern :

**What is a Saga?**

A **Saga** is a sequence of local transactions across multiple microservices. Each saga ensures that all these transactions either complete successfully or compensate for any failures to maintain data consistency.

**Core Responsibilities of a Saga:**

1. **Update the Current Microservice:**
   * Make necessary changes within the microservice.
2. **Publish Events:**
   * Trigger the next transaction in the sequence by sending events to other microservices.

**Handling Failures in Sagas?**

The **Saga pattern** manages transactions by coordinating a series of local transactions across different microservices. Each microservice has its own database and handles its transactions atomically to ensure consistency.

**How It Works:**

* **Sequential Execution:** The saga pattern groups local transactions and invokes them one after another.
* **Event Publishing:** After each local transaction, an event is published to trigger the next transaction in another microservice.
* **Failure Management:** If any step fails, the saga initiates compensating transactions to rollback changes in previous microservices, restoring data consistency.

**Types of Saga Implementations?**

There are two main ways to implement the Saga pattern:

1. **Choreography**
2. **Orchestration**

**1. Choreography Saga Pattern**

**Choreography** coordinates sagas by having microservices communicate through events without a central controller.

* **How It Works:**
  + Each microservice performs its local transaction.
  + After completing, it publishes an event to a message broker.
  + Other microservices listen for these events and trigger their own transactions accordingly.

**Advantages:**

* **Simplicity:** Ideal for simple workflows with few participants.
* **No Central Control:** Eliminates the need for an additional coordinating service.
* **Distributed Responsibility:** Avoids a single point of failure since each participant handles its own logic.

**Disadvantages:**

* **Complex Workflows:** Can become hard to manage as the workflow grows.
* **Potential Cyclic Dependencies:** Services might end up depending on each other’s events.
* **Testing Challenges:** Requires all services to be running simultaneously to test transactions.

**2. Orchestration Saga Pattern**

**Orchestration** uses a centralized controller (orchestrator) to manage and coordinate the saga.

* **How It Works:**
  + The orchestrator directs each microservice on what transactions to perform.
  + It manages the state of each task and handles failures by triggering compensating transactions when needed.

**Advantages:**

* **Handles Complexity:** Suitable for complex workflows with many participants.
* **Centralized Control:** Easier to manage and understand the flow of activities.
* **No Cyclic Dependencies:** The orchestrator manages dependencies, preventing cycles.
* **Clear Separation of Concerns:** Participants focus only on their tasks without needing to know about others.

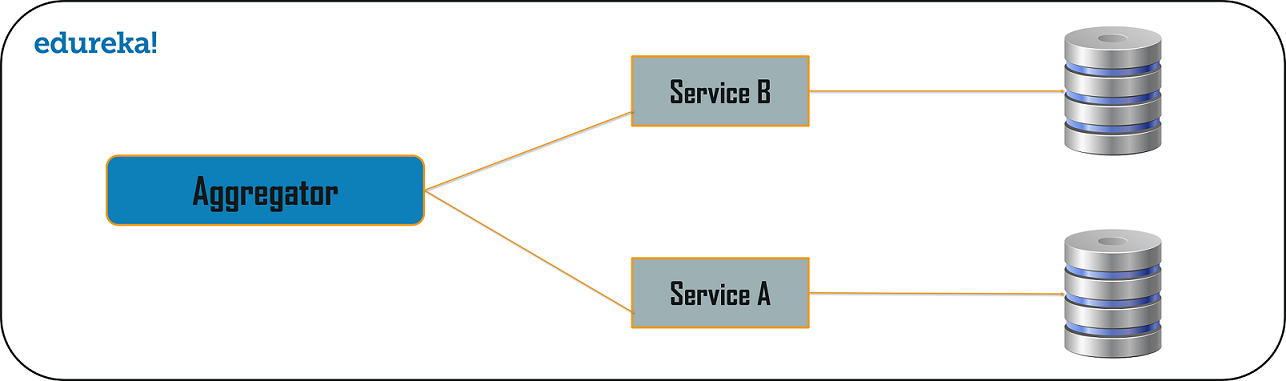
**Disadvantages:**

* **Increased Complexity:** Requires building and maintaining the orchestrator logic.
* **Single Point of Failure:** The orchestrator becomes a critical component that, if it fails, can disrupt the entire workflow.

### **Aggregator Pattern :**

In this One microservice can invokes various other microservices to get the required information or achieve the required functionality.

this pattern proves to be beneficial when you need an output by combining data from multiple services



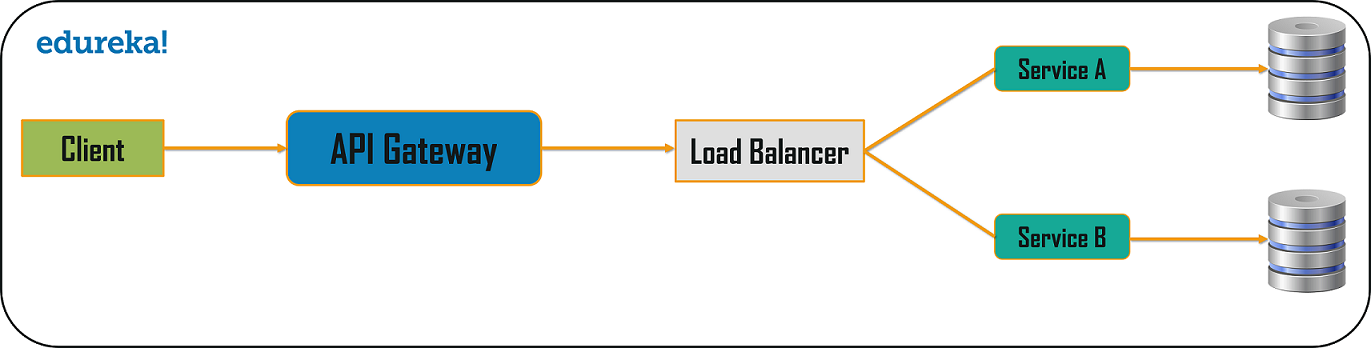
### **API Gateway Design Pattern :**

This design pattern considered as the proxy service to route a request to the concerned microservice.

In microservices application is broken down into small autonomous services

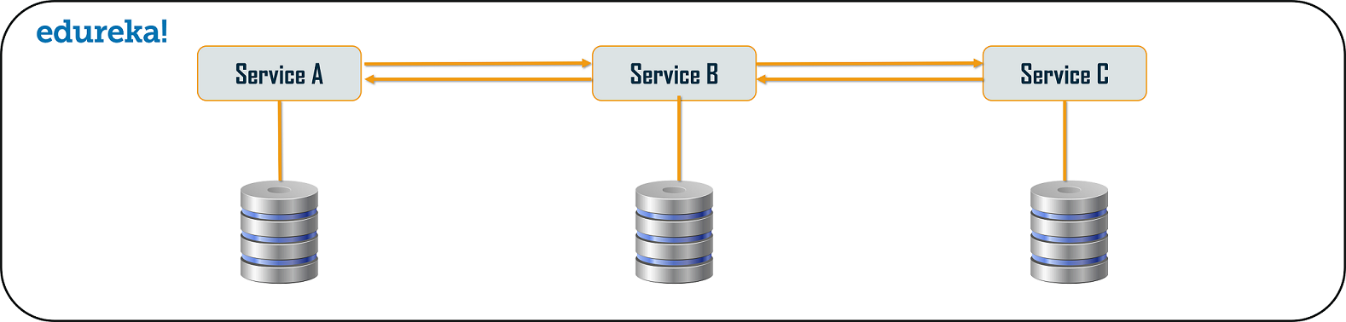
but It will cause problems like How can I request information from multiple microservices?

So to solve this we use the API Gateway Design Pattern.



### **Chained or Chain of Responsibility Pattern :**

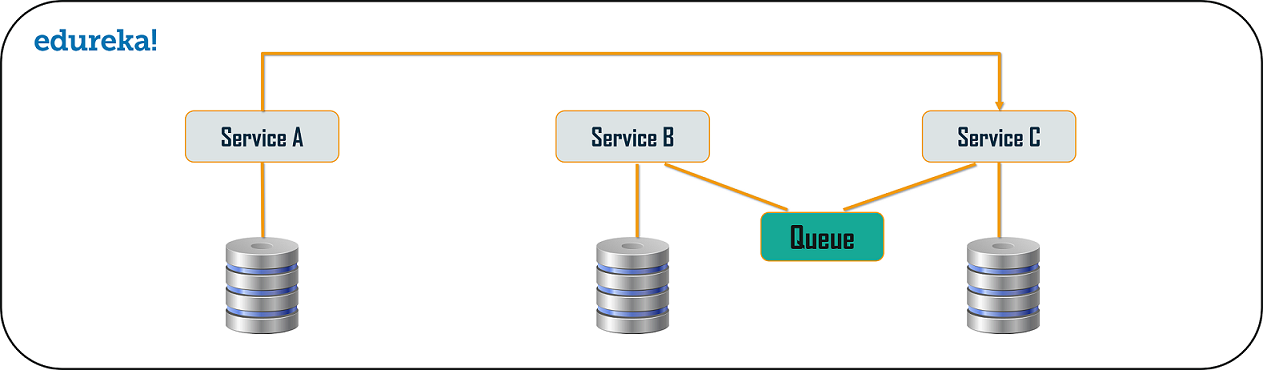
This Design Patterns produces a single output which is a combination of multiple chained outputs.



### **Asynchronous Messaging Design Pattern :**

**In Chain of Responsibility Pattern**, it is quite obvious that the client has to wait for a long time in synchronous messaging.

So to minimize this we use Asynchronous messaging.

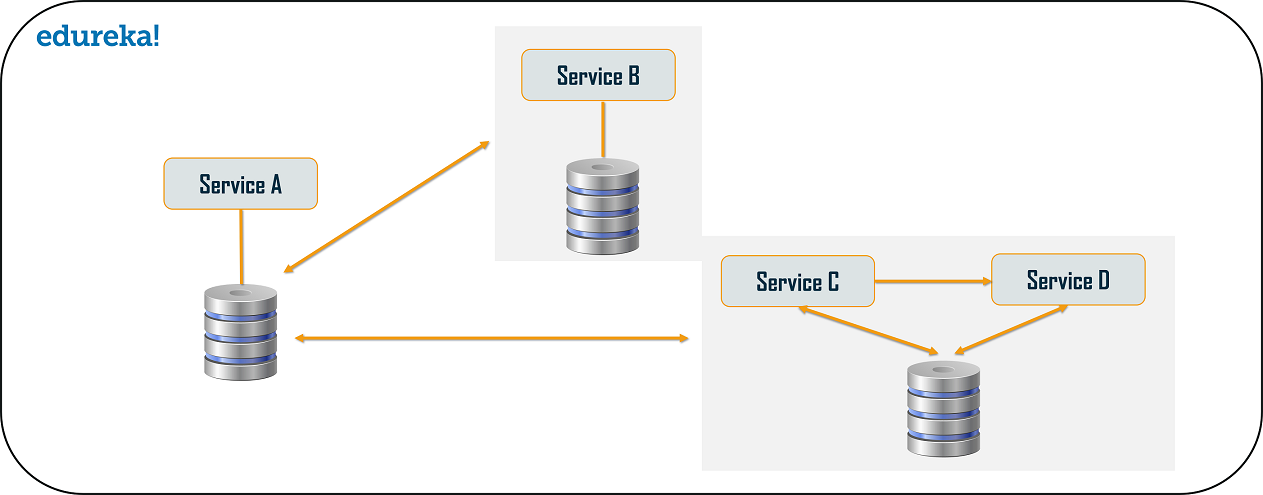


### **Database or Shared Data Pattern :**

So, when we break down an application from monolithic architecture to microservices,

it is very important to make sure each microservice has sufficient amount of data to process a request.

So to achive that we can use database per service and shared database per service to solve provide sufficient storage.



### **Command Query Responsibility Segregator (CQRS) Design Pattern :**

According to this pattern, the application will be divided into two parts:

Command and Query.

The command part will handle all the requests related to CREATE, UPDATE, DELETE

while the query part will take care of the data retriving operations.

### **Circuit Breaker Pattern :**

is used to stop the process of request and response if a service is not working.

## horizontal v/s vertical scaling :

**Horizontal scaling means that you scale by adding more machines** into your pool of resources whereas **Vertical scaling means that you scale by adding more power (CPU, RAM) to an existing machine**.

## Spring cloud :

is a set of tools that used by developers to quickly build some common patterns in distributed systems

it consist of :

1. Service registration and discovery  
2. Routing  
3. Service to service calls  
4. load balancing and circuit breakers

## Load balancing :

is a technique used to improve the distribution of workloads across several computing resources.

Spring cloud use Netflix Ribbon for this.

## **Service registration and discovery?** we have "properties" file in project.but As the project proceeds and more services are developed and deployed, adding them to the properties file becomes complex task.

Like It can affect the already deployed services

It can create problems such as services going down or the location of some might change.

So instead of Changing these properties manually Service registration and discovery are useful in such situations. To those changes can be handled by service registration and discovery.

## spring cloud bus :

it provides a helpful feature to refresh configurations across several multiple instances.

## **Hystrix circuit breaker :** Circuit Breaker is a technique, where we stop executing an erroneous method and redirect every request to a custom method (Fallback method).

Generally, we stop execution of a particular method if it is continuously throwing an exception.

## resilience4j circuit breaker :

### circuit breaker states :

1. **close : when circuit breaker is in close loop everything in working fine and all the requests are allowed**
2. **open : when circuit is open no requests are allowed**
3. **half open : while going from open to close state it goes into half open state to make sure it is working fine**

**for implementing resilience4j :**

**we have to add dependency and configure yml file**

* **business exception not be considerd as failure**

**retry : how many retry you should do when there is failure in our service and after that fallback method will be called**

**we also setup time machenism here so after that fall back method kick in**

**rate limiter : it is for to identify or to check in your system how many request should be allowd in time duration**

**like application can handle only 1000 request per/sec then we define that for particular url using rate limiter**

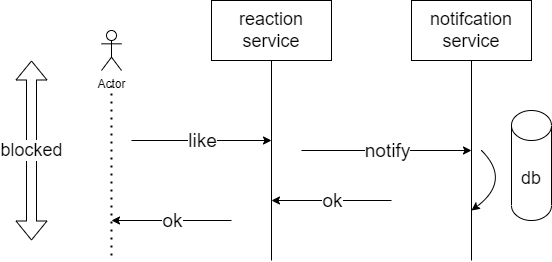
## **feign client :**

|  |  |
| --- | --- |
| Feign client | Rest template |
| 1.URLs are not hardcoded.  2.you don't have to write unit test cases for feign as there is no code to test however you have to write integration tests.  3.we can use Eureka Client ID instead of the URL.  4.Feign handled the actual code.  5.Feign integrates with Ribbon and Eureka Automatically.  6.Feign provides a very easy way to call RESTful services. | 1. **RestTemplate** is used for making the synchronous call. 2. When using RestTemplate, the URL parameter is constructed programmatically, 3. and data is sent across to the other service. 4. In more complex scenarios, ***we will have to get to the details of the HTTP APIs provided by RestTemplate or even to APIs at a much lower level***. |

## Synchronous communication :

It is simplest way of establishing communication between two microservice

In this request made from one service to another service is blocking



In this main api is blocked until all the synchronous communication is done

But this will guarantee completeness

This communication happens with the help of Rest like we expose certain end ponints and and another service gets that end points

Main adavantage is communication happenes in realtime and it is quite simple

Disadvantages :

caller is blocked until the response is recived, it would be issue if it takes to much time

servers need to be ready for peak situations

we need active load balancing and server scaleing

risk of cascading failure :

like we have chain of services a--🡪b--🡪c-🡪 d (x)goes down

it takes time to know c that d is down

it would create pressure on c , it would might chock up

we can can use circuit breaker to mitigate this but is would add additional pressure on services and services and dercease performace and goes down

tight coupling between microservices

like one should be aware of changes happening in one service

when to use :

when we can not move on(we need result before moving forword)

ex. Db queries,api response

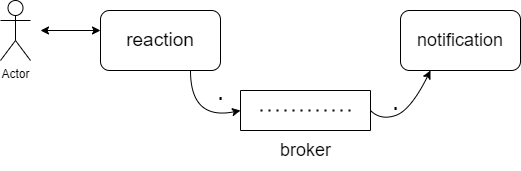
when we want realtime response

ex. Chat application,payment checkout

when it will take relitavely less time to compute and response

## ASynchronous communication :

When two services need to talk to each other instead of directly invoking each other api they send out a msg via broker (this is buffer that hold message together) so that another service can slowly consume that msg.



In this reaction service send msg broker (like user a is reacted to post by b) and this is read by notification service and notication service would say(“user a is reacted to post by b so I need to send notification to b”) and notification will sent to b

In this broker is working as a giant buffer where all the reaction sent to broker and notification will picking them one by one

So if the consuming service is down , msgs will be consumed when service comes backs so no cascading failure

In this request made from one service to another service is non-blocking

Advantages :

Service don’t need to wait for response

System can handle surge and spike better

In case of serge msgs will be pile up in broker but no change in user experience

Consumer service would be need to scale up to handle back log

No load balancer required so no extra delay

No request loss because of msg buffer

Better control over failure :

In case of failure you can always retry because msg is there

Decoupled services

Disadantages :

Eveneventual consistency :

We can not have strongly consistant system with broker hence we have to be ok with messages eventually will be consumed

But with brokers our systems does scale better

Broker is very imp :

As broker is backbone of our system we has to super cautious about that

Harder to track flow of communication :

Because of hybrid communication of microservices with each other

Need good monitoring for that

When to use :

When delay in processing is ok

Ex. Notification ,reporting

When job in hand is long running

Tracking services ,taking backups

When multipal services react to same events

Ex. When we publish blog on website

There are 3 independent task needs to work after this

1. Index in search
2. Notify followers
3. Update user analaytics

Like one msg would be consumed by three services

When it is ok for you to allow failure and retry

Ex. send notification if not send retry

## Synchronous vs Asynchronous

|  |  |
| --- | --- |
| Synchronous | Asynchoronous |
| in this model one microservice sends a request to another microservice and wait for the response.  Synchronous basically means that you can only execute one thing at a time.  Ex.  if we want to download a file from the internet in synchronous way but in that it will block all other operations until the file finished downloading.  This can make your application unresponsive to any user input. | In this model the microservice does not wait for a response,   it will send the data to another microservice and continue working independently of without waiting for response  Simply said asynchronous execution is doing stuff in the background.  Ex.  instead we could download the file in the background using asynchronous way and All the download operations are done in the background and your program will be notified when it's finished. |

**Name the services that provide service registration and discovery :**  
Eureka

zookeeper.

**What is the major difference between Spring Cloud and Spring boot?**  
Spring cloud is a microservice management framework which used to manage individual microservices

while Spring boot is used to develop these microservices.

## **Spring cloud annotations :**

@EnableConfigServer

@EunableEurekaServer

@EnableDiscoveryClient

@EnableCircuitBreaker

@HystricCommand

**Coupling**: It is defined as a relationship between software modules like how much one module depends or interacts with another one.

Couplings fall into three major categories :

highly coupled (highly dependent),

loosely coupled,

uncoupled from each other.

**Cohesion**: It is defined as a relationship between two or more elements of a module that serves the same purpose.

Generally, a module with high cohesion can perform a specific function efficiently without needing communication with any other modules.

High cohesion enhances the functionality of the module.

## **how independent microservices communicate with each other :**

rest api

or a broker or server program that uses advanced routing algorithms.

like Kafka is message brokers used in microservices where data is continuous updating without any request(ex. cricbuzz, weather apps).

## OAuth :

OAuth is (Open Authorization Protocol)

It enables users to authenticate themselves with third-party service providers.

Like we can access client applications on HTTP for third-party providers such as GitHub, Facebook, etc.

Using it, we can also share resources on one site with another site without requiring their credentials.

## **Service Discovery :**

it is one of the services running under microservices architecture,

which registers entries of all of the services running under the services network.

So whenever the services are up and running, they registers themselves to discovery service

and discovery services

also we kind of say checks the heartbeat of those services to make sure that those services are alive.

it also serves the purpose of monitoring services as well.

### Eureka :

is an application that keeps track of all client-service applications.

As every Microservice registers to Eureka Server,

Then it knows all the client applications running on the different ports and IP addresses.

It generally uses Spring Cloud and is not heavy on the application development process.

# Kafka :

so Apache Kafka is Realtime event streaming platform which works on publisher and subscriber model

where the publisher creates and sends streams of data

and at the other end subscriber process that stream of data at real-time.

so by doing so it basically makes our system more efficient.

It is also work as a message broker in a Asynchronous communication

**Benefits :**

* fault tolerance no request droping.
* scales easily without down time.
* It maintains stable performance even in peak times.

**Uses :**

* Mostly it is used as a messaging broker in Asynchronous communication between microservices.
* Kafka can be used across an organization to collect logs from multiple services and make them visible to consumers
* Also it is used in stream procressing :
* Like Streaming framework reads data from a topic, processes it and writes processed data to a new topic where it becomes available for users and applications.

## Kafka components :

1. **producer :** it is source of data who will publish messages or events.

2. **consumer :** it acts as a reciver by consuming messages or events.

3. **broker :** it is work like a server, basically it jus a intermedeator which helps in message exchange between producer and consumer.

3.1. **topic :** so it kind of acts like a category,

like if we take example of make my trip,

some people may booking filgths,

some people may booking train tickts,

and lets say some peple are booking hotels

so when these requests go to servers it can categories them into different requets so subscriber can pick them according to requirement.

in this kafka term will call them topics and we can also rename them as per our requirement.

* + 1. **Partions :** so generally kafka topics categories incoming messages, but what if there as Hugh amount of data or say request coming

in these scenario will further breakdown topics in partitions so can it can work concurrently.

and as kafka is distributed system we can assign each sever for partition for greater load balancing.

* + - 1. **Offset** : so when any new message arrives it sits in a portion where it can get any spefic number.

so basically, this number is called offset.

so if in any case there is disconnect offset help in remembering where from we start reading next.

**Consumer Group** : so we have consumer in our pub-sub model.

where consumer will read the messages form partitions

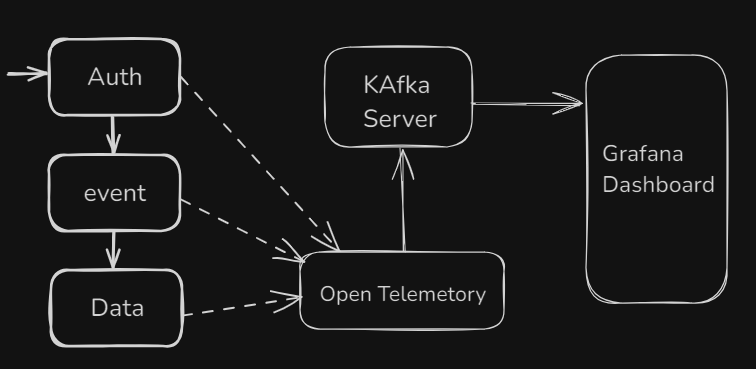
but when we are dealing with Hugh amount of data. only one consumer is not enough

so to overcome this issue and make this concurrent will group multiple consumers

where each consumer can read and process data from each partition.

**cluster :** so there is Hugh amount of data to process, cluster holds one or more broker in kafka cluster.

## How we implemented kafka :



so calypso is micro services based application.

Where we have 3 main servers :

AuthServer, EventServer, DataServer

They are co dependent on each other and we need to start them in sequence.

So what we doses is

We will start these services with openTelemetory java agent.

And then open telemetery will used to provide us observevabality

It will provide us metrics, logs, and requets data.

So we used to publish this data to Kafka server (which is fixed)

So in kafka server we have topis based on exceptions and errors.

And then we used to pull this data from kafka server on Grafana dashboard.

So this how our kafka architecture will used to work.

How to configure :  
**Step 1: configure Application.properties file**

spring.kafka.bootstrap-servers=localhost:9092

spring.kafka.consumer.group-id=your-group-id

spring.kafka.consumer.auto-offset-reset=earliest

spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer

spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer

**Step 2: Create a producer class to send messages to a Kafka topic**

@Service public

class KafkaProducer {

@Autowired

private KafkaTemplate<String, String> kafkaTemplate;

public void sendMessage(String topic, String message) {

kafkaTemplate.send(topic, message);

}

}

**Step 3: Create a consumer class to listen to messages from a Kafka topic:**

@Service

class KafkaConsumer {

@KafkaListener(topics = "your-topic-name", groupId = "your-group-id")

public void consume(String message) {

System.out.println("Received message: " + message);

}

}

**Step:4 Start kafka broker**

# Docker :

**it generally provides a container environment, in which any application can be hosted.**

**This is accomplished by tightly packaging both the application and the dependencies required to support it.**

**These packaged products are referred to as Containers, and since Docker is used to doing that, they are called Docker containers.**

**Docker, in essence, allows you to containerize your microservices and manage these microservices more easily.**

## Docker Architecture

**Dockerfile**

**Docker Image**

**Docker Registry**

**Docker Container**

## Docker Commands

**docker images : To display available docker images**

**docker pull <image-name> : download docker image**

**docker run <image-name> : creating docker container**

**docker ps : display running docker containers**

**docker ps -a : display running + stopped containers**

**docker rmi <img-id> : To delete docker image**

**docker rm <conainer-id> : To delete stopped docker container**

**docker stop <container-id> : To stop running container**

**docker start <container-id> : To re-start stopped container**

**docker logs <container-id> : To see container logs**

**docker system prune -a : to delete un-used images + stopped containers**

## Dockerfile

**It contains instructions to build image**

**We will specify application dependencies in Dockerfile**

**Naming convention : Dockerfile**

## Dockerfile Keywords

**FROM**

=> It is used to specify base image required for our application.

FROM : openjdk

FROM : tomcat8.5

FROM : mysql8.5

FROM : python-3.1

FROM : node-19

**MAINTAINER**

=> It is used to specify author of Dockerfile

MAINTAINER <Ashok@gmail.com>

**COPY**

=> It is used to copy the files from host machine to container machine

COPY <SRC> <DEST>

COPY target/app.war /usr/app/tomat/webapp.war

**RUN**

It is used to execute instructions while creating docker image

RUN 'sudo apt install git'

RUN 'sudo apt install maven'

RUN 'git clone <repo>'

**Note:** We can run write multiple RUN instructions in dockerfile and they will be processed from

top to bottom.

**CMD**

=> It is used to execute instructions while creating docker container

CMD 'java -jar <jar-file>'

**Note:** If we write multiple CMD instructions docker will process only last CMD instruction.

**EXPOSE**

It is used to specify container port number

EXPOSE 8080

**WORKDIR**

=> It is used to specify working directory

(path change)

WORKDIR /usr/app/

## Docker file example :

------------Dockerfile for running springboot app--------------------

FROM openjdk:11

COPY target/sbapp.jar /usr/app/

WORKDIR /usr/app

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "sbapp.jar"]

# Docker compose:

Docker Compose is a tool for defining and running multi-container Docker applications.

With Compose, we can use a YAML file to configure your application’s services.

Then, with a single command, you create and start all the services from your configuration.

By default, docker-compose expects the name of the Compose file as docker-compose.yml or docker-compose.yaml

## Commands :

**docker compose build** : to build

**docker compose up** : to run the containers

**docker compose ps** : check running containers list

# Kubernetes and Docker Compose

**It is used to automate deployment and management of applications in a containerized or in clustered environment**.

**So where Docker Compose is about creating and starting one or multiple containers,**

**Kubernetes serves more as a platform to create a network where we can orchestrate containers.**

* Resource optimization
* Self-healing of containers
* It reduces Downtimes during applications redeployment
* Auto-scaling

# Yourkit :

* YourKit Java Profiler is a tool used for analyzing and optimizing Java applications.
* It provides insights like memory usage, CPU performance, thread behavior, and more.
* We used to check heat map
* And we also used this to check method calls.

# Redis Cache :

## Cache :

It is something like in memory database.

It will only store relevant information which will useful for a request coming in future

Usecase :

* To reduce network calls
* Avoid precomputation
* Avoid load on database

Issues :

* There can be data inconsistency.

## Redis cache implementation:

* add starter-data-redis dependency
* we need to set redis host and port in application.properties file
* then we need to create bean for RedisTemplete
  + and there we need to mention sterilizer and deserilizer(StringRedisSerilizer)
* then in application we need to create instance of RedisTemplate
* then with this RedisTemplate we can call methods like
  + opsforvalue.set(key,value);
  + Object key = opsforvalue.get(key);

# Spring security :

**It used to handle lot of security vulnerabalities, because of that it is widely adopted.**

**Uses :**

**Username/password authentication**

**Application level authorization**

**Intra application authorization like Oath**

**Microservices security (using tokesns , jwt)**

**Method level security**

**5 core concepts in spring security :**

## **Authentication :**

**It is commonly done by providing username and password**

**It is also called knowledge based authentication**

**Big advantage is it is simple**

## **Authorization:**

**So everytime anyone ask to access anything in application the application will do binary decision should we allow it or not this is what authorization is**

**It also has some functionality to stop anyone from doing anything in application**

**For authorization we has to need authentication first**

## **Principle :**

**It is currently logged in user**

**Application remember it and we do not have to logged in again and again**

## **Authority :**

**These are bunch of permission allow for the user , in spring boot it is also called granted authority**

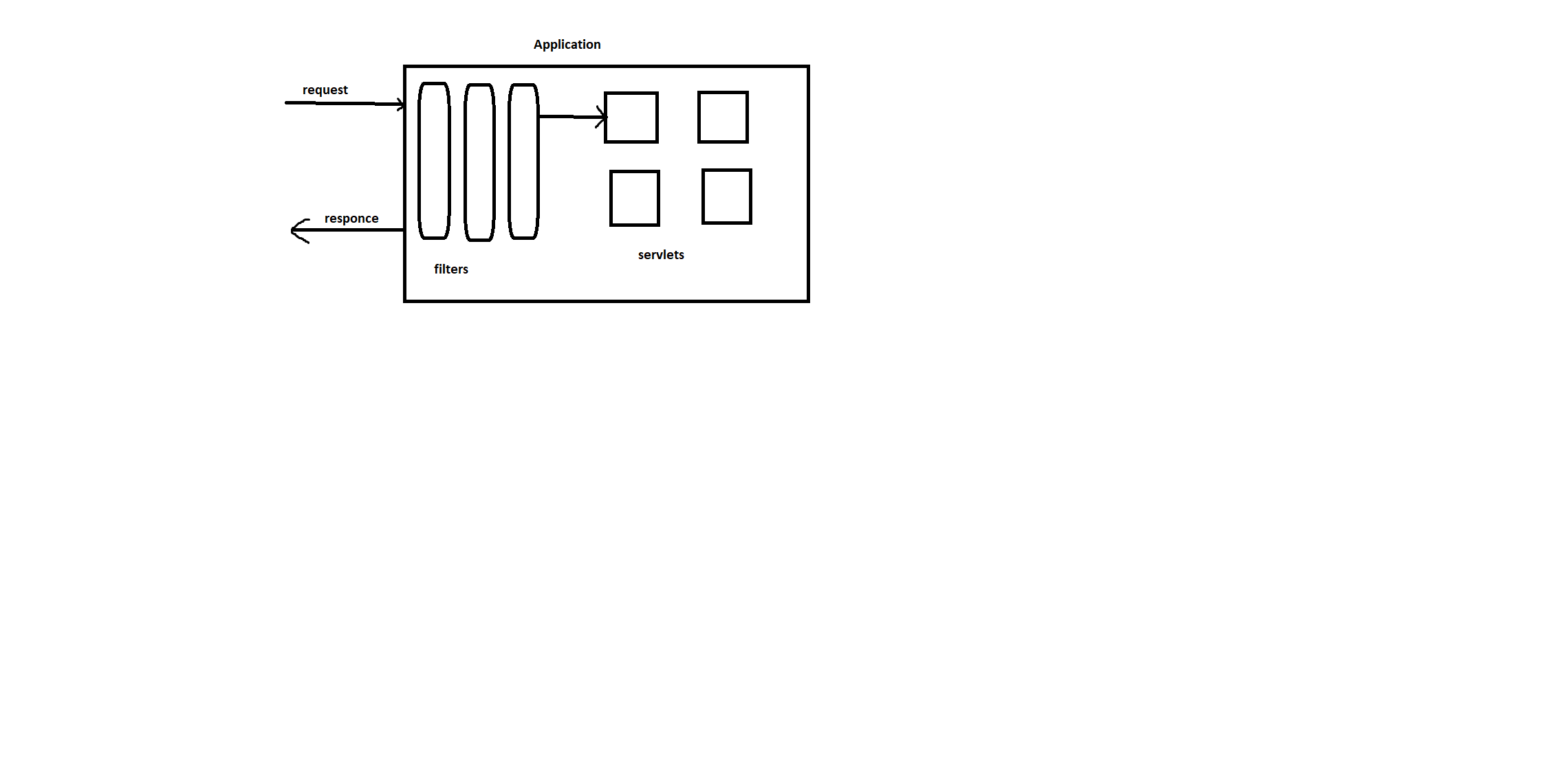
## **Roles :**

**It is same as granted authority ,some api uses it interchangeably**

## **How Spring security works :**

**Spring security adds itself in request processing by using filters**

**(filters are basically intercept any request)**

****

**Authentication : it is an interface in spring security and object of this holds the input and after details**

****

****

# Jwt Authentication :

JSON Web Tokens (JWT) is a JSON-encoded representation of a claims that can be transferred between two parties.

A JWT is a mechanism to verify the owner of some JSON data.

So when a server receives a JWT,

it can guarantee the data it contains can be trusted

because it’s signed by the source. No middleman can modify a JWT once it’s sent.

Use :

A very common use for JWT is as an API authentication mechanism.

How to expire a single JWT token :

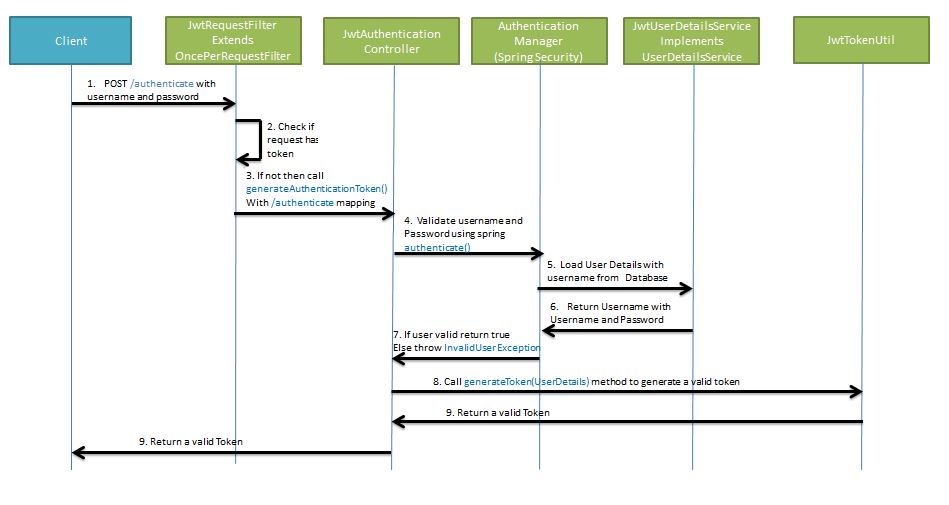
We have to add a property to user object in the server database, to reference the date and time at which the token was created.

A token automatically stores this value in the iat property

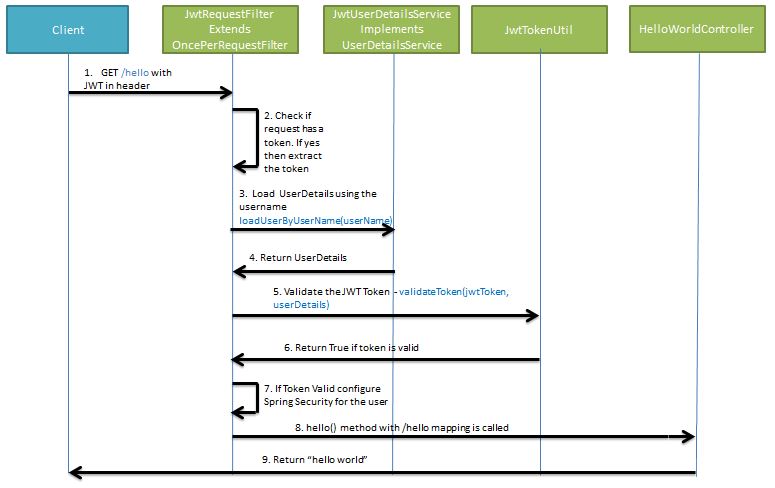
Every time you check the token, you can compare its iat value with the server-side user property.

So To invalidate the token, just update the server-side value. If iat is older than this, you can reject the token.

**Generating JWT**



**Validating JWT**

****

## Steps :

1. add json web token dependency
2. create class jwt authentication\_entry\_point implement it with Authentication\_entry\_point
   1. override all methods
   2. public void commence() it called when unauthorize user exception occuers
3. create class jwtTokenHelper : (copy paste class)
4. create class AuthenticationFilter extend with OncePerRequestFilter
5. create class jwtAuthResponce
6. configure jwt in SecurityConfig Class

# Sql

**Constrains : means to apply restrictions on particular column.**

Column Level Constraint is used to apply a constraint on a single column.

Table Level Constraint is used to apply a constraint on multiple columns.

## Constraints available In my sql :

[NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) – it Ensures that a column cannot have a NULL value

[UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different

[PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE.

[FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Prevents actions that would destroy links between two tables

[CHECK](https://www.w3schools.com/sql/sql_check.asp) – it Ensures that the values in a column satisfies a specific condition

[DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified

### Primary vs unique

|  |  |
| --- | --- |
| Primary key | Unique key |
| In case of primary key we can not save null values  Each table can holds just one column with primary key.  With the primary key, we cannot modify or delete the values. | In this we can save null values  A table can hold more than one column with unique key.  With the unique key, we cannot modify or delete the values. |

Both are enforceing uniqueness In particular column.

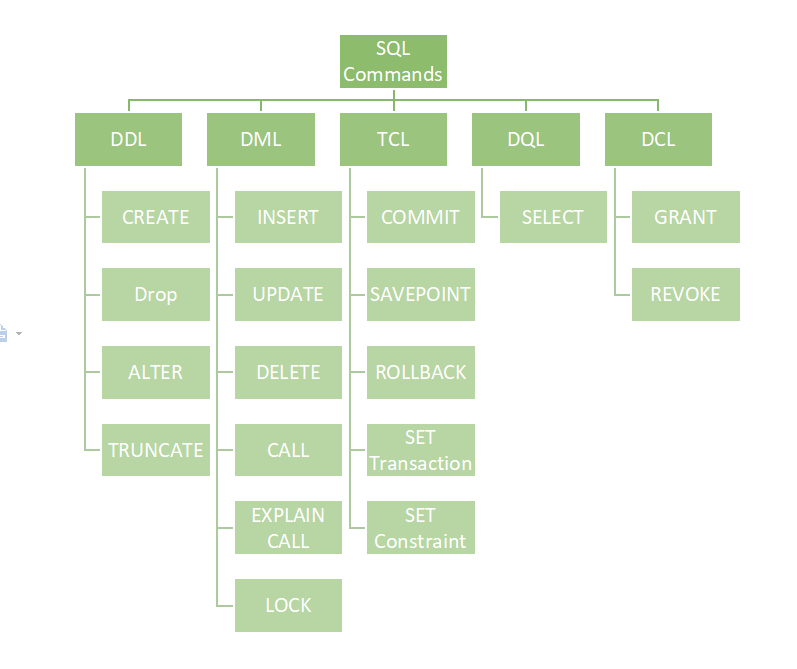
DDL – Data Definition Language

DML – Data Manipulation Language

TCL – Transaction control language

DQL – Data Query Language

DCL – Data Control Language



### Ddl vs dml

|  |  |
| --- | --- |
| **DDL** | **DML** |
| DDL stands for Data Definition Language. | Where dml stands for Data Manipulation Language. |
| DDL used to create database schema and can be used to define some constraints as well. | Where dml is used to add, retrieve or update the data. |
| ddl basically defines the column of the table. | Where dml add or update the row of the table. |
| DDL consist of :  CREATE, DROP, RENAME, ALTER etc. | DDL consist of :  UPDATE, INSERT, MERGE etc. |

### Truncate vs drop vs delete

|  |  |  |
| --- | --- | --- |
| **Truncate** | **Drop** | **Delete** |
| It is part of DDL  It is used to delete all the rows in one go.  By using this command the existence of all the rows of the table is lost.  It faster than the delete command .  Here we can’t restore the table by using the “ROLLBACK” command. | It is part of DDL  It is used to drop the whole table.  With the help of the “DROP” command we can (delete) the whole structure in one go  By using Drop command the existence of the whole table is lost.  Here we can’t restore the table by using the “ROLLBACK” command because it auto commits. | It is part of DML  It is used to delete one or more tuples of a table.  With the help of the “DELETE” command, we can either delete all the rows in one go or can delete rows one by one.  Here we can use the “ROLLBACK” command to restore the tuple because it does not auto-commit. |

A COMMIT means that the changes made in the current transaction are made permanent and become visible to other sessions.

A ROLLBACK statement, on the other hand, cancels all modifications made by the current transaction.

## Join v/s union

| **JOIN** | **UNION** |
| --- | --- |
| JOIN combines data from many tables based on a matched condition between them | union combines the result-set of two or more SELECT statements. |
| It combines data into new columns. | It combines data into new rows |
| Number of columns selected from each table may not be same. | Number of columns selected from each table should be same. |
| Datatypes of columns selected from each table can be different. | Datatypes selected from each table should be same. |

## joins

|  |  |  |
| --- | --- | --- |
| **Natural Join** | **Equi Join** | **Inner Join** |
| It joins the tables based on the same column names and their data types. | It joins the tables based on the equality or matching column values in the associated tables. | It joins the tables based on the column name specified in the ON clause explicitly. It returns only those rows that exist in both tables. |
| It always returns unique columns in the result set. | It can return all attributes of both tables along with duplicate columns that match the join condition. | It returns all the attributes of both tables along with duplicate columns that match the ON clause condition. |
| The syntax of a natural join is given below:  SELECT [column\_names | \*] FROM table\_name1 NATURAL JOIN table\_name2; | The syntax of equijoin is given below: SELECT column\_name (s)  FROM table\_name1, table\_name2, ...., table\_nameN WHERE table\_name1.column\_name = table\_name2.column\_name; | The syntax of inner join is given below: SELECT [column\_names | \*]  FROM table\_name1  INNER JOIN table\_name2 ON table\_name1.column\_name = table\_name2.column\_name; |

**Self-Join :**

self-join as two identical tables. But we cannot create two copies of the table so you just simulate having two tables with self-join.

We can use this to check duplicate records

Ex.

SELECT A.id FROM studentdb A, studentdb B

WHERE A.name = B.name

AND A.age = B.age

AND A.id != B.id

## Clause :

Order By Clause : when we want to sort data in specefic order

Select \* from emp order by salary DASC/ASC;

Group By Clause : it combines all those records that have identical value in particular field

We can also use aggregate function here

Select stream, count (\*) from student group by stream

Having clause : it is used to apply condition on group by

Select stream, count (\*) from student group by stream having count(\*)>2

Where clause : it is used to apply condition on single row

Select \* from emp where id = ‘10’;

Sub queries :

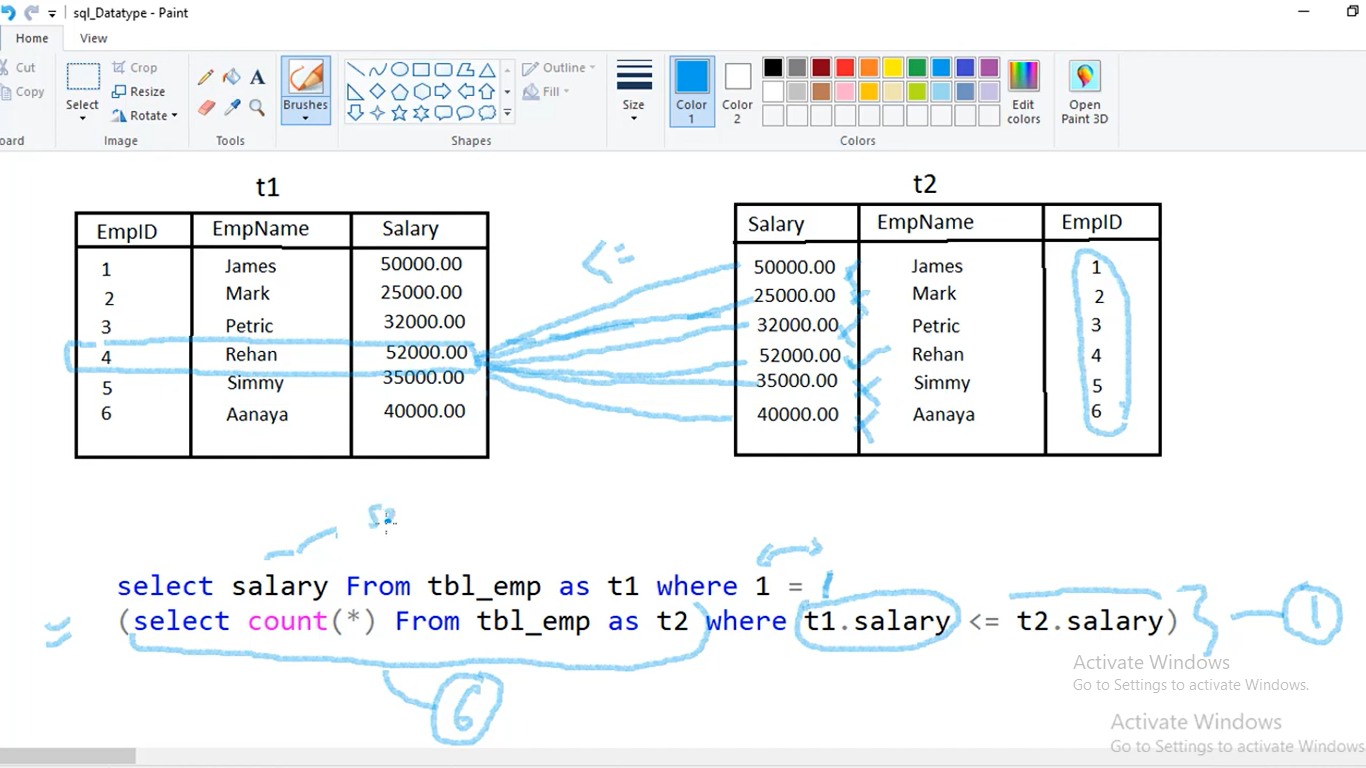
## Q. display employee whose salary is less than sid salary ?

Select salary from employees where salary < (select salary from employee where emp\_name = “sid”);

## Q.find second max salary from employees ?

Select MAX(salary ) from employees where salary < (select max salary from employees)

## Q.find nth max salary from employees ?



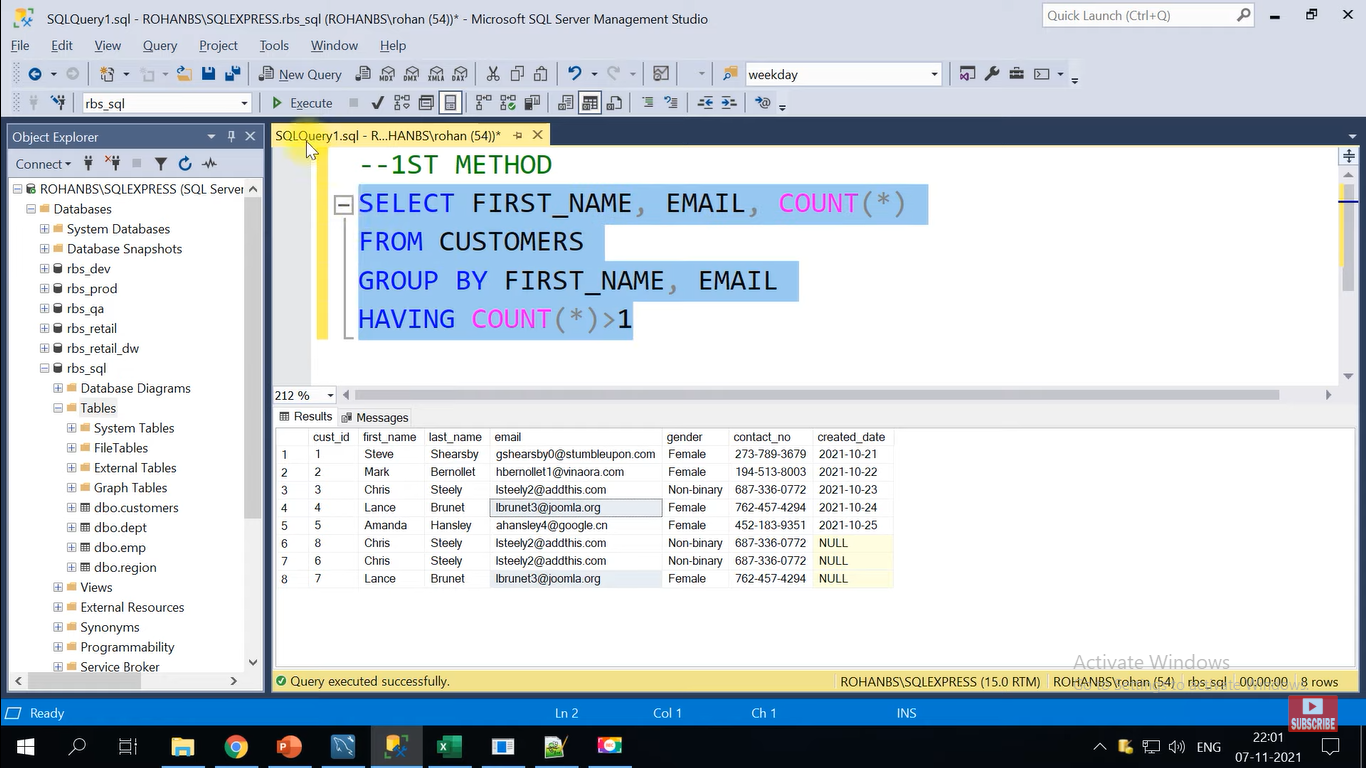
1. nested query is going to execute
2. select count(\*) from tbl\_emp as t2 == it will return no. of records 6
3. after that where t1.salary column <= t2.salary (it will compare salary of t1salary with t2salary)
4. 1 means me talking about first salary here and checking with all the salaries in t2 table
5. If it is <= This will return true
6. If this is >= it will return false
7. In this way it checks all the conditions
8. In end this will return

{=(select count(\*) from tbl\_emp as t2 where t1.salary column <= t2.salary )}

it will count of all true conditions in this case 2

then it will check 1==2 if it is false it will check until it gets true with all records

## Q. find duplicate records ?



## Indexing :

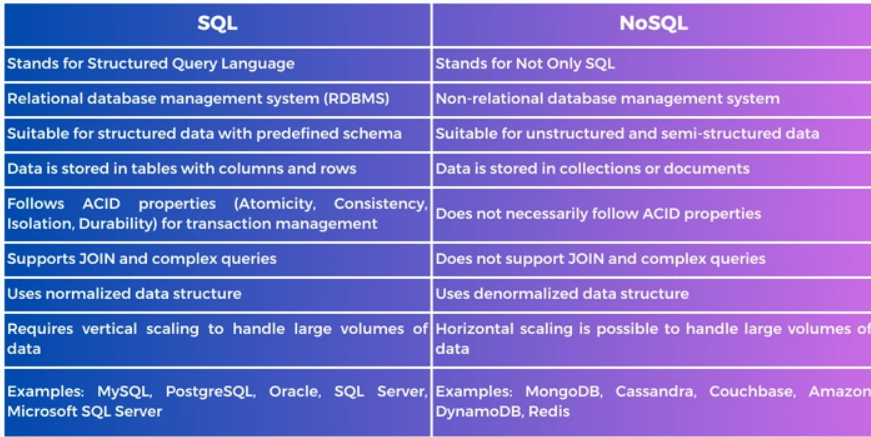
It is used to make searching fast in database

Unique and primary key columns are by default in indexes

It is mostly used in join column situation



# Sql vs no SQl :



# NoSQl databases:

## Advantages :

# Junit :

We write tests to verify the correctness of a program's behaviour.

junit is Java-based unit testing framework.

it plays a crucial role in (Test Driven Development) environment.

Important Features of Junit Testing -

It is open source testing framework allowing users to write and run test cases effectively.

Provides various types of annotations to identify test methods.

It is very simple and hence saves time.

It gives test case results in simple and elegant way.

You can integrate jUnit with Eclipse, Android Studio, Maven & Ant, Gradle and Jenkins

## Is it mandatory to write test cases for every logic?

There are no strict rules in this area, only guidelines. You usually have one test case per class, but there are different strategies:

[TestCase Per Class](http://xunitpatterns.com/Testcase%20Class%20per%20Class.html)

[TestCase Per Feature](http://xunitpatterns.com/Testcase%20Class%20per%20Feature.html)

[TestCase Per Method](http://xunitpatterns.com/Testcase%20Class%20per%20Feature.html#Testcase%20Class%20Per%20Method)

[TestCase Per User Story](http://xunitpatterns.com/Testcase%20Class%20per%20Feature.html#Testcase%20Class%20Per%20User%20Story)

## Annotations :

@Test : it indicates that method to which it is attached can be executed as a test Case.

@Before : This annotation is used if we want to execute some statement such as preconditions before each test case.

BeforeClass : This annotation is used if we want to execute some statements before all the test cases for e.g. test connection must be executed before all the test cases.

@After : This annotation is used if we want to execute some statement such as postconditions after each test case.

Eg : resetting variables, deleting temporary files ,variables, etc.

@AfterClass : This annotation can be used if you want to execute some statements after all test cases for e.g. Releasing resources after executing all test cases.

@ignores : This annotation can be used if you want to ignore some statements during test execution for e.g. disabling some test cases during test execution.

@Test(timeout = 500) : it used when we want to set some timeout during test execution e.g. if there is requirement tests need to be completed within some specified time.

@Test(expected=IllegalArgumentException.class) : This annotation can be used if you want to handle some exception during test execution. e.g., if we want to check whether a particular method is throwing specified exception or not.

@RepeatedTest : it indicates that the test method will be run n numbers of times.

JUnit methods are designed to return void, any other return type will cause exception.

test suite : it is a bundle of multiple unit test cases which can be run together.

We can use @RunWith and @Suite annotations over the test class for running the test suite.

## Assert Methods :

These belong to the Assert class in JUnit 4 and the Assertions class in JUnit 5. It is recommended to import the assert methods statically to the test class for avoiding using the class as a prefix to the method.

assertEquals() :

This method compares 2 objects for equality

it internally uses equals() method of the object.

When two objects are found to be equal then assertEquals() returns normally. Else, an exception will be thrown and the test will stop its execution.

Ex. assertEquals(50, multiplier.multiply(10, 5));

assertTrue() :

This method tests whether the value of a variable is true

Ex . @Test

public void checkNumTest() {

int num = -20;

assertFalse("Number is less then 0",num<0);

}

assertFalse() :

This method tests whether the value of a variable is false.

Ex. @Test

public void checkNumTest() {

int num1 = -20;

assertFalse("Number is not greater than 0",num1>0);

}

assertNull() :

This method tests if a variable is null.

Ex. @Test

public void checkNotNullTest() {

int num1 = null;

assertNotNull("Number is null",num1);

}

assertNotNull() : This method tests whether a variable is not null. If it is not null, then the test returns normally else an exception will be thrown and the test stops its execution.

@Test

public void checkNotNullTest() {

int num1 = null;

assertNotNull("Number is null",num1);

}

assertSame() : This method checks if two references of the object are pointing to the same object.

@Test

public void checkAssertSameTest() {

Object num1 = new Object();

Object num2 = new Object();

assertSame(num1, num2);

}

If the object references are pointing to the same object, the test runs normally else, it will throw an exception and the test execution is aborted.

assertNotSame() : This method checks if two references of an object are not pointing to the same object.

@Test

public void checkAssertSameTest() {

Object num1 = new Object();

Object num2 = new Object();

assertSame(num1, num2);

}

If the object references are not pointing to the same object, the test runs normally else, it will throw an exception and the test execution is aborted.

## How does JUnit help in achieving tests isolation?

For calling test methods, JUnit creates separate individual instances of the test class.

For example, if the test class contains 10 tests, then JUnit creates 10 instances of the class to execute the test methods.

In this way, every test is isolated from the other.

## How to ignore tests in JUnit?

In JUnit 4, we can achieve this by using @Ignore annotation over the test methods.

In JUnit 5, we can do it using @Disabled annotation over the test methods.

## How can we test protected methods?

For this test class should be declared in the same package as that of the target class.

## Define code coverage? What are the different types of code coverages?

There are three types of coverage techniques :

Questions Statement coverage: This ensures that each statement/line in the source code is executed and tested.

Decision coverage: This ensures that every decision point that results in true or false is executed and run.

Path coverage: This ensures that every possible route from a given point is run and tested.

How can we do testing for private methods?

It is generally not required to test private methods directly. Since they are private.

We use reflection api for that

MyObject obj = new MyObject();

Method privateMethod = MyObject.class.getDeclaredMethod("getFoo", null);

//reflection api contains **Method**

privateMethod.setAccessible(true);

String returnValue = (String) privateMethod.invoke(obj, null);

System.out.println("returnValue = " + returnValue);

## Testing Service Layer



## Testing Repo Layer :



## Positive and Negative Unit testing :

**Positive Testing** - testing the system by providing valid data.

**Negative Testing** - testing the system by providing invalid data.

Ex.

like there is one service method that should only accept Strings.

So By providing only String as input data to that method and check whether its working properly or not means it is Positive Testing.

If giving the input other than String means it is negative Testing..

Negative testing improves the testing coverage of your application.

Using the negative and positive testing approaches together allows you to test your applications with any possible input data (both valid and invalid) and can help you make your application more stable and reliable.

# Mockito :

It is Java-based, mocking framework

that allows the creation of test objects that simulate the behaviour (mock) of real-world objects.

Features :

we don’t need to write mock objects on your own.

It Supports return values, exceptions also Supports check on order of method calls.

It has support of annotations

## thenReturn() v/s doReturn() :

thenReturn and doReturn are used for setting up mock results

They are generally used along with when clause

doReturn takes Object parameter, Hence there is no type check in doReturn at compile time.

In the case of thenReturn , whenever the type mismatches during runtime, the exception is raised.

This is why thenReturn is a better option whenever we know the type.

Ex. when(studentobj.getRollNo(s1)).thenReturn(12);

## How to mock void method ?

doNothing-when : This is used when we do not want to check for the return parameters and skip the actual execution.

When this mocked method is called, then it does nothing.

@Test

public void updateItemTest() {

Item item = new Item(2, "Item 1");

doNothing().when(itemRepository).updateItem(any(Item.class));

itemService.updateItem(item);

}

# GitHub :

## **Local :**

Pull from remote repo to local repo main

$ git clone <https://github.com/siddheshm67/StoreManagerApp.git>

Create new branch

$ git branch -c branch\_name

Switch from current branch to another branch

$ git checkout branch\_name

\*\*Make changes\*\*

Add those changes to staging view

$ git add .

Commit changes to local repo

$ git commit -m "comment"

Push to branch to remote repo

$ git push -u origin branch\_name

Generate PR request(code review)

Merge to main

## Merge conflict :

It happens when git is unable to automatically resolve differences between two git commits

It can only automatically merge changes when commits are on different line or branches

**Solution :**

1. **Actually we don’t get into merge conflicts often**
2. **And if get We use rebase command**
3. **Like we take latest pull and commit those changes and we sort conflicts manually .**

Make changes

Add those changes to staging view

$ git add .

Commit changes to local repo

$ git commit -m "comment"

## **Remote :**

$ git push <https://github.com/siddheshm67/StoreManagerApp.git>

Then create pr request to merge with main branch

Important Terminology :

Branch- repository diverges from main working directory.

Checkout- checkout is used for the act of switching between different versions of a target entity

Clone: making copy from server.

Merge – combining branches

Origin- remote repository from a project was initially cloned

Pull- receive the data from Server (GITHUB)

Push- Upload local repository to sever.

Git Ignore-use for intentionally untrack the file

Git Diff- shows changes between commit, working tree etc.

Git Rm- for removing file

Other commands :

Let start

Create a local repository:

$ git init

Make copy

$ git clone

Adding file to staging area

$ git add file //single file

$ git add -A //all files

See the status of file

$ git st

Committing the change

$ git commit -m “comment”

Record the file permanently

Track the changes that have not been staged

$git diff

Track the changes that have staged but not committed

$git diff –-staged

Track the changes after committing a file:

$git diff HEAD

Show the objects

$ git show

Commit History

Display the most recent commits and status of the head.

$git log

$git log -p -2

Output as one commit per line

$git log –-oneline

Display the files that have been modified

$git log –-stat

Display the modification on each line of a file:

$ git blame <file name>

Ignoring Files

Git ignore file

Branching

List a branch

$git branch –-list

Create Branch

$git branch [name]

Delete Branch

$git branch -d [name]

Renaming the branch

$git branch -m [old name] [new name]

Git checkout

Switch between branch in a repository

$git checkout [branch name]

Create new branch and switch to it

$git checkout -b [branch name]

Merging

Merge the branches

$git merge [branch name]

Working on Remote

$git remote -v

Add remote to repository

$git remote add [name] [remote url]

Remove from Delete the file

$git rm [file]

Only remove file from staging area

$git rm –-cached [file]

# Maven :

It is a project management tool

* Maven install : when we call this, it internally performs one build life cycle which consist of
* It validate the project like all necessary information is available or not
* Then it compile the source code of the project
* After that it test the compiled source code ( using testing framework. )
* Then it packages code in desiriable format,(such as a JAR.)
* Then it install the package into the local repository
* Maven clean : When we to build project

,we have to make sure we should clean all the cached output

Ex.

If we rename a class, the previous compiled version will remain in target until we run clean.

(it is harmless actually , but it could cause issues if it is autodetected by classpath scanning and the like.

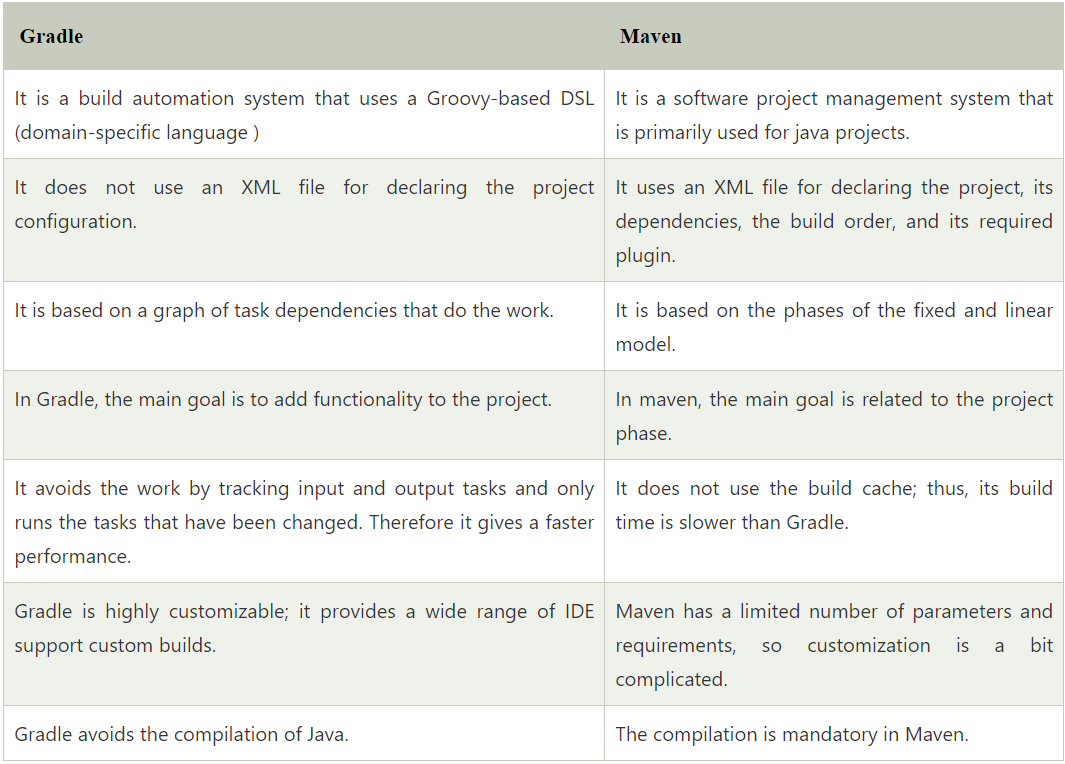
* mvn clean package : Clears the target directory and Builds the project and packages the resulting JAR file into the target directory.
* -Dmaven.test.skip=true : without running the unit tests during the build.

# Gradle :

Gradlew clean build :

Gradlew clean build -x test :

# Gradle vs maven



# jenkins:

**Deployment : in this we have to install project on client server by ourself**

**Delivery : in this we will give project to client after QA assigneing of and client has his own it team who will do deployment activity.**

**Jenkins : it is deployment tool developed by java**

**We will generate .War file in end from build tools**

**QA sign of : means QA says ok to your code**

**We have to install plugins in Jenkins to perform jobs**

## Do you have conducted any deployment activity ?

We have to deploy through jenkins

We have code in centralized repository

We have maven plugin in Jenkins

Devops team has created jobs as per our requirement

# Current task :

I have to write data from mysql database to csv file format

Using Spring Batch :

1. Create shipmentInfo entity class
2. Then I have to add one mapper which will map data returned from the brokers table to the broker entity.
3. So I created shipmentInfoMapper.java which will implement the RowMapper interface.

@Component

public class shipmentInfoMapper implements RowMapper<ShipmentInfo> {

@Override

public ShipmentInfo mapRow(ResultSet resultSet, int i) throws SQLException {

shipmentInfo sinfo=new shipmentInfo ();

shipmentInfo.setId(resultSet.getLong("id"));

shipmentInfo.setroute(resultSet.getString("route"));

shipmentInfo.setDeliveryDate (resultSet.getDate("date"));

shipmentInfo.setBrokerId(resultSet.getLong("id"));

and areacode shipment state deliverystate etc…

Return shipmentInfo ;

}

}

Then I have to create BatchConfig file :

Like it is a class where we will configure the spring batch job with JdbcCursorItemReader  to read data from the database

and FlatFileItemWriter to create CSV files and write data in CSV files,

also we will be using FlatFileHeaderCallback to add header in CSV files.

# Programs :

## Anagram:

**public** **static** **void** main(String[] args) {

String string1 = "LISTEN";

String string2 = "siting";

String lowerCase1 = string1.toLowerCase();

String lowerCase2 = string2.toLowerCase();

**char**[] charArray1 = lowerCase1.toCharArray();

**char**[] charArray2 = lowerCase2.toCharArray();

HashMap<Character, Integer> hashMap1 = **new** HashMap<>();

HashMap<Character, Integer> hashMap2 = **new** HashMap<>();

**for** (**char** c : charArray1) {

**if** (hashMap1.get(c)== **null**) {

hashMap1.put(c, 1);

}**else** {

hashMap1.put(c, hashMap1.get(c)+1);

}

}

System.***out***.println(hashMap1);

**for** (**char** c : charArray2) {

**if** (hashMap2.get(c)== **null**) {

hashMap2.put(c, 1);

}**else** {

hashMap2.put(c, hashMap2.get(c)+1);

}

}

System.***out***.println(hashMap2);

**boolean** equals = hashMap1.equals(hashMap2);

System.***out***.println(equals);

## Find occurrence :

**public** **class** Note {

**public** **static** **void** main(String[] args) {

String string = "javaitiseasytolearn";

String[] arrstr = string.split("");

Map<String, Integer> hMap = **new** HashMap<>();

**for** (String c : arrstr) {

**if** (hMap.get(c) == **null**) {

hMap.put(c, 1);

}**else** {

hMap.put(c, hMap.get(c)+1);

}

}

System.***out***.println(hMap);

}

}

## Find nth higest/lowest occuring elements :

String string = "....";

String[] arrstr = string.split("");

Map<String, Integer> hMap = **new** HashMap<>();

**for** (String word : arrstr) {

**if** (hMap.get(word) == **null**) {

hMap.put(word, 1);

}**else** {

hMap.put(word, hMap.get(word)+1);

}

}

Map<Integer, String> map = **new** TreeMap<>();

//convert map to treemap

hMap.forEach((e1,e2)->map.put(e2, e1));

System.***out***.println(map);

//find higest/lowest occuerence

Entry<Integer, String> entry =

map.entrySet()

.stream()

.sorted((e1,e2)->-e1.getKey().compareTo(e2.getKey()))

.skip(0)

.findFirst().get();

//create new arraylist to store result element

List<String> l2= **new** ArrayList<>();

**for** (String string2 : arrstr) {

l2.add(string2);

}

Set<String> set = l2.stream()

.filter(e->Collections.*frequency*(l2, e)==entry.getKey())

.collect(Collectors.*toSet*());

System.***out***.println(set);

## Find higest occurring element using java 8 :

String input = "abcccgfdrsaaabcck";

// Step 1: Convert String to a Stream of Characters

Map<Character, Long> charCountMap =

input.chars()

.mapToObj(c -> (char) c) // Convert each int to a Character

.collect(

Collectors.groupingBy(Function.identity(), Collectors.counting())

); // Count occurrences of each character

// Step 2: Find the Character with the Maximum Count

Character mostFrequentChar = charCountMap.entrySet()

.stream()

.max(Comparator.comparingLong(Map.Entry::getValue)) // Find the entry with the maximum count

.get() // Get the result (Optional)

.getKey(); // Extract the character (key)

System.out.println(mostFrequentChar);

## Find repeating elements :

**public** **class** Note {

**public** **static** **void** main(String[] args) {

String string = "javaitiseasytolearn";

String[] arrstr = string.split("");

List<String> list = **new** ArrayList<>();

**for** (String c : arrstr) {

list.add(c);

}

Set<String> set = list.stream()

.filter(e->Collections.*frequency*(list,e)>1)

.collect(Collectors.*toSet*());

System.***out***.println(set);

}

## Find nth repeating elements :

String string = "javaitiseasytojlearn";

String[] arrstr = string.split("");

List<String> list = **new** ArrayList<>();

**for** (String c : arrstr) {

list.add(c);

}

Set<String> set = **new** LinkedHashSet<>();

list.stream()

.filter(e->Collections.*frequency*(list, e)>1)

.forEach(e->set.add(e));

Optional<String> nth = set.stream().skip(nth - 1).findFirst();

System.***out***.println(nth.get());

## Find non-repeating elements :

List<String> l = list.stream()

.filter(e->Collections.*frequency*(list,e)==1)

.collect(Collectors.*toList*());

## Find nth non-repeating elements :

Optional<String> nth = list.stream()

.filter(e->Collections.*frequency*(list, e)==1)

.skip(nth-1).findFirst();

## Check occurrence by using array :

**public** **class** CheckOccuerance2 {

**public** **static** **void** main(String[] args) {

String string = "jpareposetory";

**char**[] charArr = string.toCharArray();

**int** b[] = **new** **int**[charArr.length];

**for** (**char** c : charArr) {

System.***out***.print(c+" ");

}

**for**(**int** i = 0;i<b.length;i++) {

b[i]=-1;

}

**for** (**int** i = 0; i < charArr.length; i++) {

**int** count = 1;

**for** (**int** j = i+1; j < b.length; j++) {

**if** (charArr[i]==charArr[j]) {

count++;

b[j] = 0;

}

}

**if** (b[i] != 0) {

b[i] = count;

}

}

**for** (**int** i = 0; i < b.length; i++) {

**if** (b[i]>1) {

System.***out***.println(charArr[i]+" : "+b[i]);

}

}

}

}

## Bubble Sort (double loop):

**public** **class** OptimizedBubbleSort **extends** Bbb {

**public** **static** **void** main(String[] args) {

**int** arr[] = {3,2,1,5,8,9};

**for** (**int** i = 0; i < arr.length; i++) {

**boolean** flag = **false**;

**for** (**int** j = i; j < arr.length; j++) {

**if** (arr[i]>arr[j]) {

**int** temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

flag = **true**;

}

}

**if** (flag == **false**) {

**break**;

}

}

System.***out***.println(Arrays.*toString*(arr));

}

## Bubble Sort (Single loop):

**public** **static** **void** main(String[] args) {

**int** arr[] = { 3, 2, 1, 5, 8, 9 };

**for** (**int** j = 0; j < arr.length - 1; j++) {

// Checking the condition for two

// simultaneous elements of the array

**if** (arr[j] > arr[j + 1]) {

// Swapping the elements.

**int** temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

// updating the value of j = -1

// so after getting updated for j++

// in the loop it becomes 0 and

// the loop begins from the start.

j = -1;

}

}

System.***out***.println(Arrays.*toString*(arr));

}

## Check Palindrome number:

**public** **class** REverseNumber {

**public** **static** **void** main(String[] args) {

**int** num = 214;

**int** rem =0;

**int** rev = 0;

**while**(num!=0) {

rem = num%10;

System.***out***.println(rem);

rev = rev \*10+rem;

num = num /10;

}

System.***out***.println(rev);

**if**(num == rev) {

System.***out***.println("yes");

}**else** {

System.***out***.println("no");

}

}

}

## Check Palindrome String:

**static** **boolean** checkWord(String str) {

String wordOne = "";

**for**(**int** i = str.length()-1;i>=0;i--) {

wordOne = wordOne + str.charAt(i);

}

**if** (str.equals(wordOne)) {

**return** **true**;

}**else** {

**return** **false**;

}

}

## Common elements in Arrays :

**public** **class** CommonELementInArray {

**public** **static** **void** main(String argString[]) {

**int** a[] = {1,2,5,7,9};

**int** b[] = {1,5,9};

**int** c[] = {1,4,5,7,9};

**int** x = 0;

**int** y = 0;

**int** z = 0;

List< Integer> list = **new** ArrayList<>();

**while**(x<a.length && y<b.length && z<c.length) {

**if** (a[x] == b[y] && b[y]==c[z]) {

list.add(a[x]);

x++;

y++;

z++;

}**else** **if** (a[x]<b[y]) {

x++;

}**else** **if** (b[y]<c[z]) {

y++;

}**else** {

z++;

}

}

}

}

## Factorial :

**public** **class** Factorial {

**public** **static** **void** main(String[] args) {

**int** fact = 1;

**for**(**int** i = 1;i<=5;i++) {

fact = fact \* i;

}

System.***out***.println(fact);

}

}

## FiboNachi :

**public** **class** Fibo {

**public** **static** **void** main(String args[]) {

**int** first = 0;

**int** Second = 1;

**int** sum = 0;

**for**(**int** i = 0;i<=10;i++) {

sum = first +Second;

System.***out***.print(sum+" ");

first = Second;

Second = sum ;

}

}

}

## Pattern :

**public** **class** Pattern {

\*\*\*\*

\*\*\*

\*\*

\*

**public** **static** **void** main(String []args) {

**for**(**int** i = 1;i<=5;i++) {

**for**(**int** j = 1;j<=i;j++) {

System.***out***.print(" ");

}

**for**(**int** k = 5-1;k>=i;k--) {

System.***out***.print("\*");

}

System.***out***.println(" ");

}

}

}

## PrimeNumber (1 to n):

**public** **class** PrimeNumber {

**public** **static** **void** main(String []args) {

**for** (**int** j = 2; j <=n; j++) {

**int** num = j;

**boolean** flag = **false**;

**for**(**int** i = 2;i<=Math.*sqrt*(num);i++) {

**int** rem = num%i;

**if** (rem==0) {

flag = **true**;

**break**;

}

}

**if** (flag == **false**) {

System.***out***.println(j);

}

}

}

}

## Check Prime :

**public** **static** **void** main(String[] args) {

**int** num =7;

**boolean** flag = **false**;

**for**(**int** i = 2;i<=Math.*sqrt*(num);i++) {

**int** rem = num%i;

**if** (rem==0) {

flag = **true**;

**break**;

}

}

**if** (flag == **true**) {

System.***out***.println("not an prime number");

}**else** {

System.***out***.println("prime number");

}

}

}

## Reverse each word :

**public** **static** **void** main(String[] args) {

String s1 = "my name is siddhesh";

String resultString ="";

String[] split = s1.split(" ");

**for** (String string : split) {

String s2 = "";

**for**(**int** i = string.length()-1;i>=0;i--) {

s2 = s2 + string.charAt(i);

}

resultString = resultString.concat(s2).concat(" ");

}

System.***out***.println(resultString);

}

## Swap String without 3rd variable :

**public** **static** **void** main(String args[])

    {

        // Declare two strings

        String a = "xyz";

        String b = "abc";

        // Print String before swapping

        System.out.println(“a : "+a+ “ b = "+b);

        // append 2nd string to 1st

        a = a + b;

        // store initial string a in string b

        b = a.substring(0,a.length()-b.length());

        // store initial string b in string a

        a = a.substring(b.length());

        // print String after swapping

        System.out.println(“a : "+a+ “ b = "+b);

    }

}

## Move Zero to Right :

**public** **class** TestClass121 {

**public** **static** **void** main(String[] args) {

**int**[] arr = {1,5,6,0,4,0,3,2,9,7,0};

**int** length = arr.length;

**int** count = 0;

**for** (**int** i = 0; i < arr.length; i++) {

**if** (arr[i]!=0) {

arr[count++] = arr[i];

}

}

**while** (count<length) {

arr[count++] = 0;

}

**for** (**int** i : arr) {

System.***out***.print(i+" ");

}

}

}

## Move Zero to left :

**public** **class** TestClass121 {

**public** **static** **void** main(String[] args) {

**int**[] arr = {1,5,6,0,4,0,3,2,9,7,0};

**int**[] arr2 = **new** **int**[arr.length];

**int** count = 0;

**for** (**int** i = 0; i < arr.length; i++) {

**if** (arr[i]==0) {

count++;

}

}

System.***out***.println(count);

**for** (**int** i = 0; i < arr.length; i++) {

**if** (arr[i]!=0) {

arr2[count++] = arr[i];

}

}

**for** (**int** i : arr2) {

System.***out***.print(i+" ");

}

}

}

## immutable class :

// class is declared final

final class Immutable {

// private class members

private String name;

private int date;

Immutable(String name, int date) {

// class members are initialized using constructor

this.name = name;

this.date = date;

}

// getter method returns the copy of class members

public String getName() {

return name;

}

public int getDate() {

return date;

}

}

class Main {

public static void main(String[] args) {

// create object of Immutable

Immutable obj = new Immutable("Programiz", 2011);

System.out.println("Name: " + obj.getName());

System.out.println("Date: " + obj.getDate());

}

}

## Removing Consecutive Duplicate Words

**public** **static** **void** main(String[] args) {

String input = "He is Programmer Programmer . He is is Teacher.";

String[] words = input.split(" ");

StringBuilder output = **new** StringBuilder();

String prevWord = words[0];

output.append(prevWord);

**for** (**int** i = 1; i < words.length; i++) {

String word = words[i];

**if** (!word.equals(prevWord)) {

output.append(" ");

output.append(word);

prevWord = word;

}

}

System.***out***.println(output.toString());

}

# Excel report generation:

We have used Apache POI for this

public static void main(String[] args) throws Exception {

// workbook object

XSSFWorkbook workbook = new XSSFWorkbook();

// spreadsheet object

XSSFSheet spreadsheet = workbook.createSheet(" Emp Data");

// creating a row object

XSSFRow row;

// This data needs to be written (Object[])

Map<String, Object[]> studentData = new TreeMap<String, Object[]>();

studentData.put( "1", new Object[] { "Emp No", "NAME", "Salary" });

studentData.put( "1", new Object[] { "1", "xyz", "10000" });

studentData.put( "1", new Object[] { "2", "abc", "11000" });

studentData.put( "1", new Object[] { "3", "lmn", "12000" });

Set<String> keyid = studentData.keySet();

int rowid = 0;

// writing the data into the sheets...

for (String key : keyid) {

row = spreadsheet.createRow(rowid++);

Object[] objectArr = studentData.get(key);

int cellid = 0;

for (Object obj : objectArr) {

Cell cell = row.createCell(cellid++);

cell.setCellValue((String)obj);

}

}

// .xlsx is the format for Excel Sheets...

// writing the workbook into the file...

FileOutputStream out = new FileOutputStream(

new File("C:/savedexcel/GFGsheet.xlsx"));

workbook.write(out);

out.close();

}