**JAVA 1.8 Features:**

**Functional Interface:**

The Interface contains only one abstarct method but can have multiple default methods and static methods is called functional inteface.

ex:

Runnable

Callable

Comparable

Comparator

Public Interface MyFunctionalInterface

{

public abstract void m1();

default void m1(){ }

default void m2(){}

}

**predefined Functional Interfaces:**

Predicate

Function

Consumer

Suppiler

**two argument predefined functional interface:**

BiPredicate

BiFunction

BiConsumer

**Predefined primitive Function interfaces:**

IntPredicate

IntFunction

IntConsumer

above interfaces are available in java.util.packages:

**Predicate:**

conditional check

public abstract boolean test( T t); --> it will take one input and return boolean value.

Ex1

Predicate<Integer> p=a->a%2==0

p.test(20);

Ex2

String s[]={"amara","simha","simha1","simha2","simha3"}

Predicate<String> p= s->s.length()>5;

for(String s1:s)

{

if(p.test()){System.out.println(s1);}

}

ex3:

p1 ---> to check whether the number is even or not.

p2 ---> to check wheather the number >10 or not.

p1.and(p2).test(34);

**Predicate Joining:**

p1.or p2() --> wheather first condition is satisfied or second condition is satisfied

p1.negate(); opposite of p1

**Function:**

i have some requirement.. i have one input and perform some operation and produce some result then we are going to use Function.

Interface Function<T,R>

{

public R apply(T t)

}

ex : pass the String and produce the length of the string :

Function<String,Integer> f=(name)->name.length();

f.apply("Simhachalam");

**Function Chaining:**

f1.andThen(f2).apply(i); --> first f1 apply and then f2

f1.compose(f2).apply(i) --> first f2 apply and then f1

Ex:

Function<Integer,Integer> f1=i->2\*i;

Function<Integer,Integer> f2=i->i\*i\*i;

System.out.println(f1.andthen(f2).apply(2)); output 64

System.out.println(f1.compose(f2).apply(2)); output : 16

**Consumer:**

it will takes input and return void.

Consumer<T>

Examples:

Consumer<String> c=s->System.out.println(s);

c.accept("Simhachalam");

**Suppiler:**

Just supply the my required Objects and it wont take any input -> Supplier

interafce Supplier<R>{

public R get(){}

}

**Note: all the above functional interafce takes only one input argument and return values**

**suppose if we need two input values then we can use below functional intefaces:**

**BiPredicate**

BiPredicate is exactly same as Predicate except that it will take 2 input arguments.

inteface BiPredicate<T1 t1, T2 t2>

{  
public boolean test(T1 t1,T2 t2){}

}

**BiFunction**

public

BiFunction<Integer,String,Employee> emp=(empNo,name)->new Employee(empNo,name)

Employee e1=emp.apply(100,"Simhachalam");

**BiConsumer**

public BiConsumer<Employee,Double> emp=(emp,d)->emp.getSalary()+d;

it will take two input parameter and return void. so that it will consume.

------------------------------------------------------------------------------------------------------------------------

Predicate<Integer> p=(i)->1%2==0;

p.test(20);

here iam providing the int (Primitive values to the prdicate function) but predicate function accepting the Integer value (Object). so that internally converting into the primitive value.

**Int -->Integer -->int**

**autoboxing and autounboxing**

so that it will give big performance problem. this case we are going to take below function insteading going predicate function

**intPredicate functional Inteface we can take for above examples.**

**if we want to use double then doublePredicate**

**Premitive Predicate types:**

IntPredicate

DoublePredicate

LongPredicate

**Premitive Function Types:**

DoubleFunction --can take input type as double and return type can be any type.

**intFunction**

**longFunction**

**DoubleToIntFunction ->**can input type as Double and return type as int

and Method conatin this function as : applyAsInt()

**DoubleToLongFunction** --> input type as Double and return type as Long

and Method contain this function as s: applyAsLong();

**IntToDoubleFunction** -->input type as int and return type as Double and method contain this function as ApplyAsDouble()

**LongToIntFunction** --> input type as Long and return type as int and method contain this function as ApplyAsInt()

**ToIntFunction** --> return type as int and input can be anything

**ToDoubleFunction** --> return type as double and input can be anything

**ToLongFunction** --> return type as Long and input type can be anything.

**toIntBiFunction** --> return type as int and input can be anything

**toLongBiFunction** --> return type as Long and input type can be anything.

**toDoubleBiFunction** --> return type as double and input can be anything

**Premitive Consume Types:**

intConsumer - > input type a input

LongConsumer ->input type a Long

DoubleConsumer ->input type a Double

**Primitive types of Supplier:**

BooleanSupplier -- getAsBoolean

intSupplier -- getAsInt

LongSupplier -- getAsLong

**BinaryOperator**.. we can use if both input and return type same then can use the Binary Operator.

BiFUnction can take two input argument and return something.

if two input and and return type same then we can use Binary Operator.

***UnaryOperator:*** we input and output always same then we can use unaryOperator.

**Lambda Expression:**

always once we write lambda expression and to call the lambda expressoin then we should call from Functional inteface.

Functional interafce caontain single abstract method

@FunctionalInterface

Interface A{  
public void m1();

}

@FunctionalInterface

interface extends A

{

}

above one is perfectly valid scenario.

no .class files are genarted by lambda expression.

**Default and Static method:**

default method can be ovveride but we can not default key word in the sub class.

we can use

public void m1()

{

}

static methods are can not be overidden.

**Collections and Stream:**

if we want to represent group of objects into single entity then we should go for collection:

if we want process the objects from collection then we can use Stream.

ArrayList al=new ArrsyList<>();

al.add(10);

al.add(30);

al.add(50);

al.add(30);

Stream stream=al.stream();

List<Integer> lst=l.stream().filter(i-i%2==0).collect(collectors.toList());

if we want to add some action on the every object in the collection then we can use the MAP.

filter(Predicate) - condition check

Map(Function) --> add some action on the every object in the collection.

Methdos:

filter()

map()

stream()

collect()

count() --> how many elements are there in your stream then we can use count.

sorted() -> by default it will sort ascending order

stream().sorted().collect(collectors.toList());

stream().sorted((i1,i2)->(i1<i2)?1:(i1>i2)?-1:0).collect(collectors.toList());

min(comparator) stream.min((i1,i2)->i1.compaeTo(i2)).get();

max(comparator) stream.min((i1,i2)->-i1.compaeTo(i2)).get();

forEcah() it is using consumer internally.

where ever group of values are there then we can happily go for Stream concepts:

streams conectpts are apply can Arrays and Collections and Strings

stream.oF()

**Optional Class:**

streams Optional class having three static methods:

**empty**

**oF**

**oFNullable**

Three aways to create optional object which i was mentioned above.

**empty** and oF and oFNullable

Optional<Object> emptyOptional=Optional.empty();

**optional.oF**(here we can pass object and String) then return particular passing object or String.

Optional<String> optional=Optional.oF(string)

if the object is not giving null value then we can use this method. otherwise we can use below which i have mentioned in the below.

**Optional.oFnullable**(customer.getEmail()) here it wont throw nullPointer exception

we can not call the get method directly if we call get method then we may get "No Such element Exception"

before that we can call

**isPresent()** --> it will give true and false if value is there then it will give true or else give false.

**orElse()** --> optional.orElse("simhachalam") --> if optional object is empty then it will return orElse method argument value.

or if value is present then it will return that value.

if we know some exception need to throw if value is not empty and null then we can use below method:

**orElseThrow()**--> it will Supplier functional Interface.

Interface

orElseGet() --> it will take supplier argument

here we are checking the value in the optional object if value is there the we can do upper case it or else we will give some default value.

oprional.map(String::upperCase()).oRElseGet(()->"Default Email......"));

Map and Reduce:

Map --> transforming data.

Reduce --> Aggregating the data.

ex: Stream [2,3,7,6,9,5]

**Map**() --> Transform Stream object to Stream on int.

**Reduce**() -> combine stream of int to produce the sum result.

**reduce**(t identity,BinaryOperator<t> accumulator);

t is initial value and accumulator is a function to combine to two values.

Integer sumresult=stream.oF(1,6,7,3,2).reduce(0,(1,2)->a+b);

Numbers.stream().reduce(Integer::sum);

Integer maxvalue=numbers.stream().reduce(0,(a,b)->a>b?a:b);

mapToDouble - it return the double of stream consisting of the results of applying the given function to the elements of this Stream.

**Map and FlatMap().**

**Map** method is used for data transformation and flatMap is used for data transformation and flatering.

its mapper function produces single value for each input value. hence it is also called one to one mapping.

map will take input as stream and return another stream.

**FlatMap** --> it will take stream of stream as input and return stream <R>.

it mapper function produces multiple values for each input value hence it is called one -To- Many Mapping.

stream.of("a","b","c","d"); all are in lower case and want to transfor to uppper case.

it is return another transformation like upper case.

--> [A,B,C,D]

Transform data from Lower case to Upper Case.

[[1,2],[3,4],[5,6],[7,8]]

above stream contains stream of multiple stream then we can use the flattering map.

and the return type is another stream like bleow.

[1,2,3,4,5,6,7,8]

here converting stream of stream into single stream.

Optional class Methods:

-----------------------------------------------

public boolean isPresent()

Return true if there is a value present, otherwise false.

Returns:

true if there is a value present, otherwise false

public void ifPresent(Consumer<? super T> consumer)

If a value is present, invoke the specified consumer with the value, otherwise do nothing.

Parameters:

consumer - block to be executed if a value is present

Throws:

NullPointerException - if value is present and consumer is null

**Date And Time Api in java 1.8 features:**

JODA Time Api.

before java 1.7 --> Date and Calendar, TimeZone

**SpringBoot With MicroServices:**

**SpringBoot With Security:**

**OAuth2.O**

different types of Grant Type and Auth flow:

Autherization Code Grant Type

Implict Grant type FLow

Resource Owner Credential Grant Type

CLient Credential Grant Type

Refresh Token Grant Type.

**Autherization Code Grant Type** we need to send below parameter to Authorization server.

client\_id

redirect\_url

scope

state

response\_type -- (code so that authorization server will understand client need to authorization code)

for token we have to send below parameter to Auth server.

code

Client Id and secret id

Grant\_ type

redirect\_uri

final give token from Auth server.

**Implict Grant Type:**  this is one clubed into single step.

client\_id it will genarate after register google and gitHub

redirect\_url :

scope

state

repsonse\_type --(token) (this is implict grant Type)

this is very less security than Authorization code.

**Resource Owner Credential Grant Type:** client and auth and resource server having in the same orginization then we can use Resource Owner Credential Grant Type

step1 : User Enter username and Password and client will go to check auth server and resource server.

client\_id and secret id

scope

UserName and password

Grant\_type : password which means resource owner credential Grant type.

Realm : here we can create env profile and client and user roles.

**Client Credentional Grant type:**

we can use between apis in the different application

here there is no involvement user and resource owner

client\_id and secreateId

scope

Grant\_type 'Client\_Credential' which means want follow Client Credentinal grant type

Realm - here we can create env profile and client and roles and users

**Refresh Token grant type:**

client\_id and secreateId

refresh\_token

scope

Grant\_type 'refresh\_token' which means want follow Client Credentinal grant type

**OpenID Connect:**

it has beenimplemented on top of OAUTH2.O framework.

access Token and ID token it will response.

KeyCLock Authorization server.. we can integrated with any social network and LDAP and saml.

keyCLock acting as Authorization server

it is open source framework and access Management product.

we need to download the ZIP file and extarct and go to bin folder and run standalone.sh

once the server is started and then we can go to chrome browser and

localhost:8080/auth/

and after that we need to create admin account and refresh same page, then we can see the admin is created.

intrenally H2 database and if we want MysQl and sql in the production then we can implement.

**Realm -->** it is space here we need create configuration here --> env

**Resource Server configuration:**

in pom.xml we need to add spring-boot-oauth2-resource-server.

we need to implement Converter interface. and we have ovveride converter method.

JWTAuthenticationConverter jwtAuthenticationConverter=new JWTAuthenticationConverter();

jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(new **KeycloakRoleConverter**())

**public** **class** **KeycloakRoleConverter implements** Converter<Jwt, Collection<GrantedAuthority>> {

@Override

**public** Collection<GrantedAuthority> convert(Jwt jwt) {

Map<String, Object> realmAccess = (Map<String, Object>) jwt.getClaims().get("realm\_access");

**if** (realmAccess == **null** || realmAccess.isEmpty()) {

**return** **new** ArrayList<>();

}

Collection<GrantedAuthority> returnValue = ((List<String>) realmAccess.get("roles"))

.stream().map(roleName -> "ROLE\_" + roleName)

.map(SimpleGrantedAuthority::**new**)

.collect(Collectors.*toList*());

**return** returnValue;

}

}

{

"access\_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJFS0hhZmFleVA4QVNxZHBsVTRPaVFneUhQalNEaS16UXFVQTh2NDJwTlZBIn0..IYtRPotIsJQbDkNgh4hsCDeo1-v2KWgz33dtykm-xrL-3vJFhVKyfgWJVSsCHKqbIjPE125AzXZ1J6NWxIOrPjmDFE\_RtrHHzNMETvWD6y6qFKZdhWGMe54ZaZXR2Qz4EEdccBqIamIKutpOEcvNcEgf4k7RHEX4nV9D-mz2GJl0HWasCqRQgsqNKm2-e25TWw8EkJFw2mVo\_-0ixX6X9ig7d42lT64v8VOGMTTLlf0y\_dykk9nMPiwC69HUiXzdSqmDNI\_g4m7rsml7IBhflNZCVDd1P8UTVyhYZla9vLq\_\_qK3TVIRUdsO2Ghzo9MDJPhQLRM3Aoz9phHeci02\_w",

"expires\_in": 300,

"refresh\_expires\_in": 0,

"token\_type": "Bearer",

"id\_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJFS0hhZmFleVA4QVNxZHBsVTRPaVFneUhQalNEaS16UXFVQTh2NDJwTlZBIn0..ZUrWkZq9dfjb3z5XLcV4tFBWXW2hgRAgz7KZ-TpEwsQBHX4v-HiUI0U4x\_gUXvCCbiooQSkcCvhN2GO0cW9mhu8k3L8YdFZxap2Bm0484SRvR-9Yq-hmpkaLUvYYFMTrzlywMIt-b3Alm2\_GQcRKMvLfuokpvuLvRcFaFqoyT8C2NogRdaKnQDydx8b7CyCEMZD9In-0Q\_FQ7QB51NX\_pkjEvWXqtz4N11RNFtDT\_8SYf54U0jeBsj0fVtMl9MUte5Jj80Pbu0RZNAtMeI72FPaVawImeYXxCF0CtY1CtP2yAxMBJFGCH66g4BXM498KaW5QrwZe1c\_EaYuBxrCgFw",

"not-before-policy": 0,

"scope": "openid profile address email"

}

above code is for accessing role from openCloak into SpringSecurity Project..

and also we need to get access token from Authorization server.

**spring.security.oauth2.resourceserver.jwt.jwk-set-uri =** [**http://localhost:8080/auth/realms/SimhaBankAPI/protocol/openid-connect/certs**](http://localhost:8080/auth/realms/SimhaBankAPI/protocol/openid-connect/certs)

below url we can findout all the api in the openCloak

<http://localhost:8080/auth/realms/SimhaBankAPI/.well-known/openid-configuration>

access token genaration url along with url we need to pass below things:

<http://localhost:8080/auth/realms/simhaBank/protocol/openid-connect/token>

client\_id and client\_secret

scope ( openId email address) -> openId scope details and email and address details

Grant\_type 'Client\_Credential' which means want follow Client Credentinal grant type

**Authorization Code Flow:**

instead of sending client secrete from UI to Auth server.

in UI we are going genarate code challange by using code verifier.

first step: code challage and client id and userName and password sending to auth serevr

step2 : verify the the credentional and cleint id

and return auth code to ui

and then UI makes another call with auth code and code\_verifier. this time auth server check the code challange with code verifier. for genarating access token

KeyCloak Server..

we can configure any social network in key clock server.

client id and secrete id we need to configure in keyclock server.

Authorization code grant type with PKCE

MicroServices:

Spring-cloud-config-server will connect to Git Repository.

and all the applications will connect to spring-cloud-config-server. for that we need to add spring cloud config client dependency to all the micro services application.

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

**spring-cloud-config-server dependency --> spring cloud config server**

**spring-cloud-config client (all the microservices ) --> here we can treated as spring cloud client.**

Micro services x **MicroServices Y MicroServicesZ**

**Spring cloud Config Server**

**application.properties file like below in the spring cloud config server**

**spring.application.name=spring-cloud-config-server**

**spring.cloud.config.server.git.uri=**file:\\\c:\\Users\\Simhachalam\\Documents\\microservices\\git-localconfig-repo

**Git Repository**

**and every microservices application need to configure below parameters in the application.properties:**

**spring.config.import=optional:configserver:http://localhost:8888**

**in the Git Repository we need to create below properties:**

**limits-services-dev.properties**

**limits-services-qa.properties**

**and above properties we can access by using env variable:**

**spring.profile.active=dev**

**spring.cloud.config.profile=qa spring cloud config server will conect to Git repository. by using application**

**by different names we can use below propertie:**

**spring.cloud.config.name="" .. here we need to property names in the git repository through spring cloud config server.**

**Spring cloud Open feign**

**Service Registory : Eureka server**

**loadBalancing will happen through the feign client.**

**if 100 microServices and logging and authentication then we can use spring cloud gateway.**

**for that we have to add spring cloud routing dependency.**

**api gateway need to connect to eureka client so that we have add below property:**

**eureka.client.serviceUrl .defaultZone=http://localhost://**

**spring.cloud.gateway.discover.client.enabled=true**

@Component

**public** **class** GlobalFilters **implements** GlobalFilter {

**private** Logger logger=org.slf4j.LoggerFactory.*getLogger*(GlobalFilters.**class**);

@Override

**public** Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {

// **TODO** Auto-generated method stub

logger.info("Path of the request received from service "+exchange.getRequest().getPath());

**return** chain.filter(exchange);

}

}

@Configuration

**public** **class** APIGatewayConfiguration {

@Bean

**public** RouteLocator gatewayRouter(RouteLocatorBuilder builder) {

**return** builder.routes().route(f -> f.path("/currency-conversionfeign/\*\*").uri("lb://CURRENCY-CONVERSION")).

route(f -> f.path("/currency-exchange/\*\*").uri("lb://CURRENCY-EXCHANGE")).

build();

}

}

RouteLocatorBuilder here we can take request and process it further.

spring cloud gateway provides cross cutting concerns :

security

monitoring/metrics

**Circuit Breaker - Resilience4j:**

suppose one microservice connect to another micro services

any one micro service down then entire services will be down.

for above problem we can use circuit Breaker .. if micro service is down then we can send some response (default response).

and also we can retry the request if some temporary problem will happen

for this one we need to dependency Resilience4j to pom.xml

@Retry(name = "simpleDemo", fallbackMethod = "dummy")

**by default it is 3 times .. it will be retry**

resilience4j.retry.instances.sample-api.maxRetryAttempts=5

resilience4j.retry.instances.sample-api.waitDuration=1s

**5 retry will happen then it will call the fallbackMethod and give the response to api.**