**Java8 Features**

1.Concise Code

2.Enabling functional Programming

**1. Lambda Expressions**

**2. Functional Interface**

**3. Default Method and Static Method**

**4. Predefined Functional Interface**

**a) Predicate :** Accepts only 1 datatype ,used to check conditions

Ex: Predicate<Integer> p = i->i%2==0; //Checks given num is even or not o/p:true

System.out.println(“Even or Not: “ +p.test(4));

**b) Function :** Accepts 1 datatype for input and another datatype for output

Ex: Function<Integer,Integer> f = i->i\*i; //square of number

System.out.println(“Even or Not: “ +p.apply(50)); //o/p: 2500

**c) Consumer**

**d) Supplier**

**5. Double Colon Operator(::) : Used for Method Reference ,Constructor Reference**

**6. Streams**

**7. Date and Time API**

**8. Optional Class**

**9. Nashorn JavaScript Engine**

**1.Lambda Expressions:**

The main objective of Lambda Expression to bring benefits of functional programming into java

**Definition : Lambda Expressions is an anonymous function with no Name ,without return type, without modifier.**

**Syntax:** **() -> {}**

{ } -> Body of Expression //It’s optional if only 1 statement is present

Ex: ( a , b) -> System.out.println(a+b);

n -> n\*n ;

n -> {return n\*n; }; //If braces used return mandatory

**2. Functional Interface :** The Interface that contain only Single Abstract Method(SAM)called Functional Interface.

-> If you want to invoke(call) Lambda Expressions, Functional Interface is mandatory.

-> Functional Interface contains only **1 abstract method** but can have **any number of default and static methods**

**-**>@FunctionalInterface – specify explicitly that it is FI using this annotation(Optional)

Runnable 🡺run() method present in Runnable Interface

Comparable 🡺compareTo()

Comparator 🡺compare()

ActionListener 🡺actionPerformed()

Callable 🡺 call()

->multiple non overriding abstract methods //Error caused if we have 2 abstarct methods

@FunctionalInterface

interface A {

public void m1();

}

@Functionalinterface

interface B extends A{

}

* For interface .class file created but if we use lambda expressions in next half..no .class file will be created for lambda

\*Comparator(Used for Sorting)uses compare method(compare 2 elements)

**->compare(Object obj1,Object obj2)**

->return type int

**-> returns -ve iff obj1 has to come before obj2**

**returns +ve iff obj1 has to come after obj2**

**returns 0 iff obj1 and obj2 are equal**

**import java.util.\*;**

**class Test**

**{**

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

**{**

**ArrayList<Integer> 1= new ArrayList<Integer>();**

**1.add(20) 5**

**1.add(10) ;**

**l.add(25) 5**

**l.add(5)5**

**system.out.println(l);**

**Comparator<Integer> c= (I1,I2)-> (I1<I2) ? -1 : (I1>12) ? 1 : 0;**

**Collections.sort(l,c);**

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

**//l.stream().forEach(System.out::println);**

**}**

**}**

\*import java.util.\*; is not applicable when we are accessing subpackage classes and interface. In that case we need to mention for each separately. Ex: import java.util.Function;

**==================================================================================**

**Consumer, Supplier & Predicate**

**Consumer takes 1 input and doesn’t return anything(void)**

void accept(T t)

**Predicate Functional Interface**

**->Used for Conditional Check(true or false)**

**Boolean test(T t);**

Based on input argument u can do conditional statement or conditional check but returns boolean

**Supplier functional Interface**

**Supplier can be used in all contexts where there is no input but an output is expected.**

* 1 abstract method **T get() ;** //return type generic
* Ex:

@Functional Interface

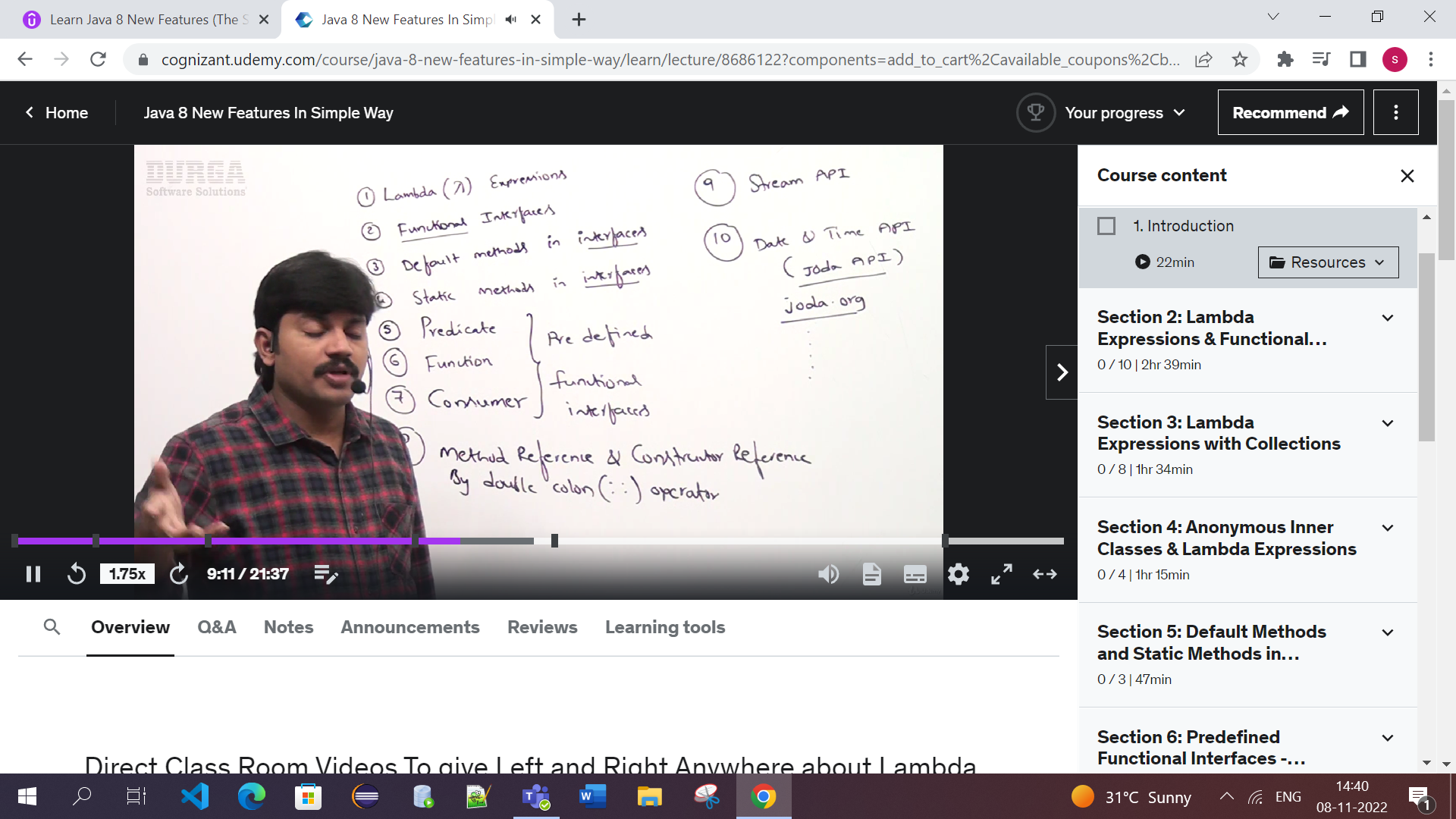
Public interface Supplier<>

{

T get();

}

Java8



Objectives of 1.8:

1.To simplify Programming

2.To utilize functional programming benefit in java

3.To enable parallel programming(run on multiple systems)

1.Lambda Expressions: It is an anonymous(nameless) function i.e., not having name,not having modifier and not having return type. Such type of anonymous functions called Lambda Expressions.

i)If body contains only 1 statement then curly braces are optional.{}

ii)If the compiler can guess data types based on the context we can remove datatype.

Ex: (a,b)->sopln(a+b)

Properties:

History:

LISP is first programming language which uses LambdaExpressions.Similarly C,c++,SCALA,RUBY.Finally Java also started using Lambda Expressions

Advantages:

1.To enable functional programming

2.To write more readable ,maintainable & concise code

3.To use API’s easily and effectively

4.to enable parallel programming

Ex:1.public void m1() ->Here remove public is modifier ,void return type,m1 method

{

SOPln(“Hello”);

}

So after removing function name,return type,modifier by using lambda expressions

() -> { SOPln(“Hello”);} -------------🡪Lambda Expressions

Ex:2 public void add(int a,intb) (a,b) ->{SOPln(a+b);}

{

Sopln(a+b);

}

Ex:3 public int getlength(String s)

{

return s.length();

}

(String S)->{return s.length();}

**s->s.length(); -🡪Simplified lambda Expression**

2.**Functional Interface**: An interface that contains only 1 abstract method.i.e., Single Abstract Method(SAM)

Ex: Runnable ->contains only run() method

ActionListener- >contains only actionPerformed()

Callable->contains only call() method

2.It is used to invoke(call) Lambda Expressions

3.We can take any number of static and default methods but only 1 abstract method is restricted.

[4.@FunctionalInterface](mailto:4.@FunctionalInterface) – If we use this annotation compiler warns at compile time if we used more than 1 abstract method Ex:Unexpected @FunctionalInterface annotation; multiple non-overriding abstract methods present in interface.

Ex:@FunctionalInterface

interface Interf

{

public void m1();

default void m1()

{}

public static void m3(){

}  
}

How to invoke lambda expressions by using Functional

Interface

Ex 1: interface Interf interface Interf

{ {

public void m1(); public void m1();

} }

Class Demo implements Interf Class Test2

{ {

public void m1(){ psvm(String args[] )

Sopln(“m1 () method implementation”); {

} Interf i = ()->sopln(“m1() method

implementation);

} i.m1();

Class Test1 }

{

Public static void main(String[] args) }

{

Interf i=new Demo();

i.m1();

}

}

Using Lambda expressions