**JAVA 8 Features**

Here are the things which we will cover under this session –

1. Understanding Lambda
2. Using Lambda
3. Functional Interface
4. Method References
5. Collections improvements

Since this is new paradigm so this is very important to understand why JAVA has brought these changes and how this is going to help developers.

One more and important thing needs to understand is, was there anything which we were not able to achieve by JAVA until JAVA7 and that will be achievable after JAVA8?

There will be more and more questions that will come to your mind and will be answered as this will progress.

Why Lambdas and Advantages –

* It enables functional programming
* Readable and concise code, removes boiler plate codes
* Easier to use APIs and Libraries
* Enables supports for parallel processing

Dev Environment setup

So before we start with the Lambda and other JAVA 8 features and concepts, lets understand

Functional Programming vs Object Oriented Programming

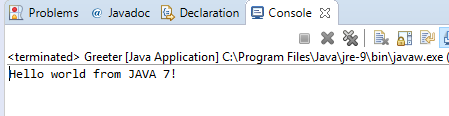
So when we talk about the Functional Programming in JAVA, it’s not going to change is existing implementation of JAVA. JAVA is an Object Oriented programming and it will remain so.

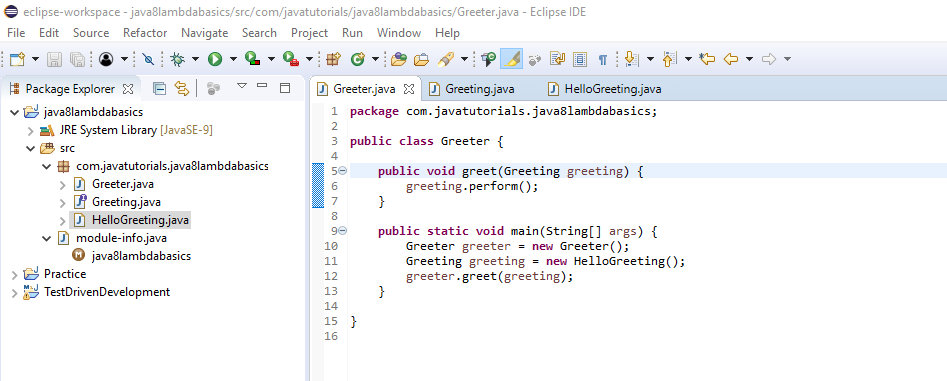
Having said that JAVA 8 supports functional programming, let’s look at how JAVA 8 solves the problems.

So let’s start with one classic example of OOPs where we want a method to perform an action –

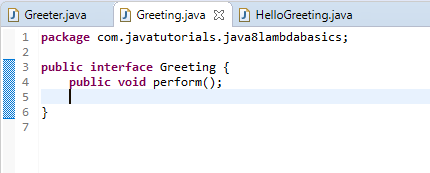
Main class – Greeter.java

Where looking for desired output as –

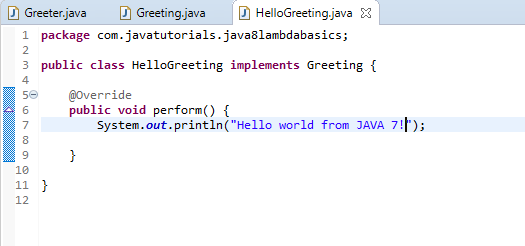




Interface for Actions – Greeting.java



Implementation class for Action – HelloGreeting.java



So as we were looking for greet method to perform an Action which is being passed, now we are doing so in above programing by passing an interface Greeting which has perform method. But if we notice we can see we are not actually passing the action instead we are passing an interface which has action and our target is to just pass an action something like

**public** **void** greet(Greeting greeting) {

greeting.perform();

}

**public** **void** greet(action) {

action();

}

Lambda allows to do so, it creates just the function which does not belongs to any class and the best part is that function can be assigned to a value.

So, what is function as a value?

First let’s understand the inline values –

String a = “Abhishek”

double d = 3.14

Now the question is can we write something like

aBloackOfCode = {

-----

}

So from above example let’s assign one function perform to a variable –

aBloackOfCode = public void perform(){

System.out.println(“Hello world from JAVA 7!”);

}

So, now when we see above example there are few things which are no point using here, as public which will be now dependent on the variable to which we are assigning, similarly the name of the method which is perform. So now it should look like –

aBloackOfCode = void (){

System.out.println(“Hello world from JAVA 7!”);

}

Now complier is smart enough to understand the return type, so remove void as well

aBloackOfCode = (){

System.out.println(“Hello world from JAVA 7!”);

}

And this will turn to Lambda like –

aBloackOfCode = () -> {

System.out.println(“Hello world from JAVA 7!”);

}

Few Lambda examples before we start, just to train our mind –

greetingFunction = () -> System.out.println(“Hello world from JAVA 7!”);

doubleNumberFunction = (int a) -> {

return a\*2;

};

doubleNumberFunction = (int a) -> a\*2;

addFunction = (int a, int b) -> a+b;

safeDivideFunction = (int a, int b) -> {

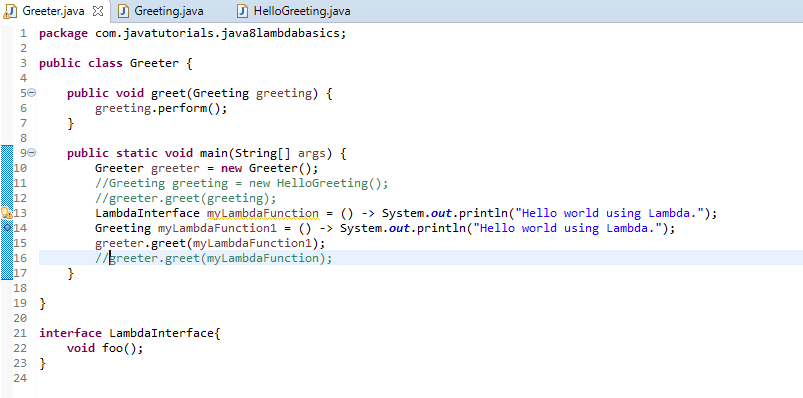
if(b==0) return 0;

return a/b;

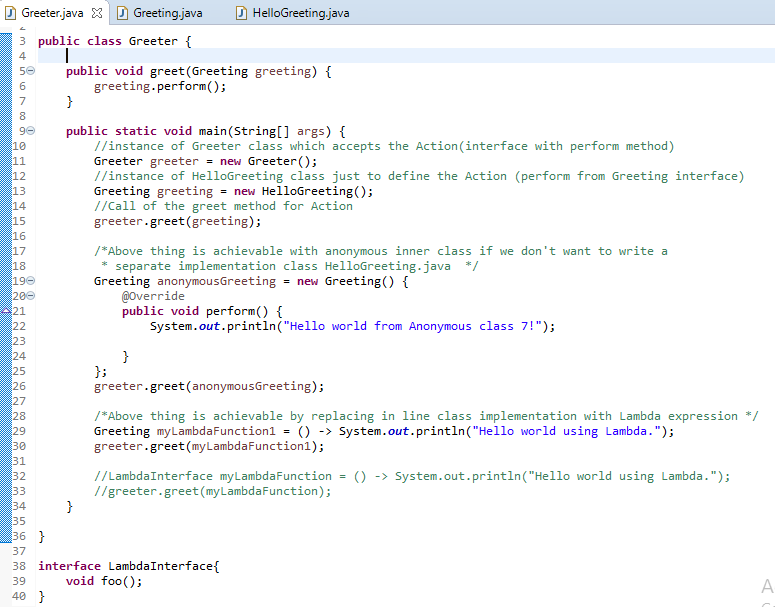
};

stringLengthCountFunction = (String s) -> s.length();

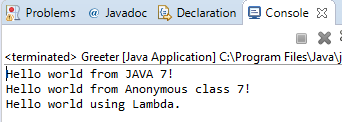
So, now let get our hands dirty with Lambda concepts –



Explanation of Lambda using programming –

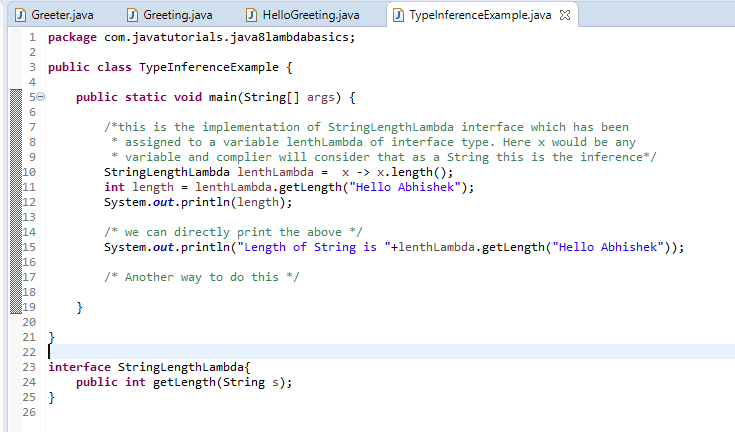


Output

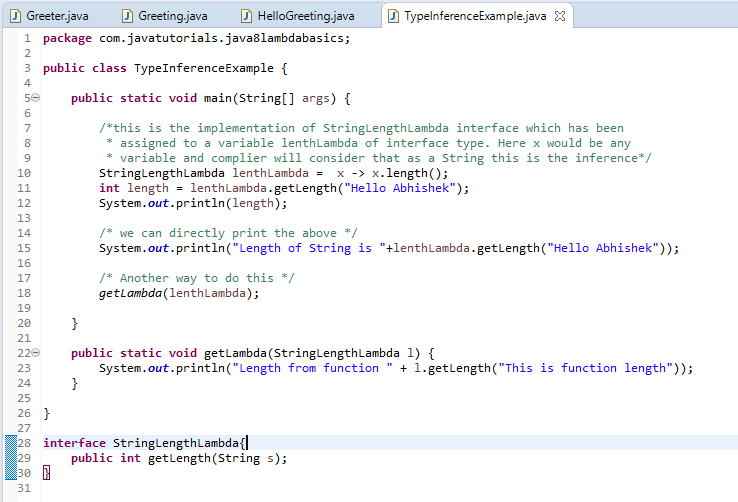


Type Inference –

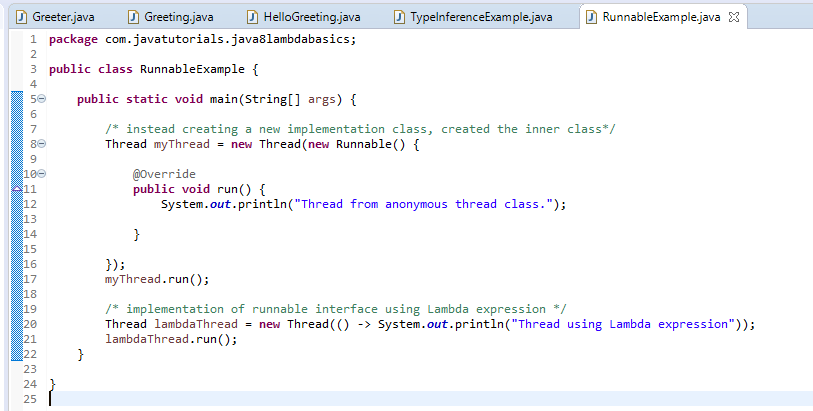
This is a contract between Lambda expression and functional interface.



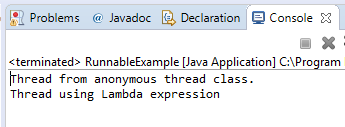
Now, as I said we can pass the Lambda expression in a function as a parameter as well. Let’s see –



One classic example, why Java did not introduce new Lambda type instead used the interface type.



Output –



Functional Interface –

It’s an interface with one abstract method.

@FunctionalInterface

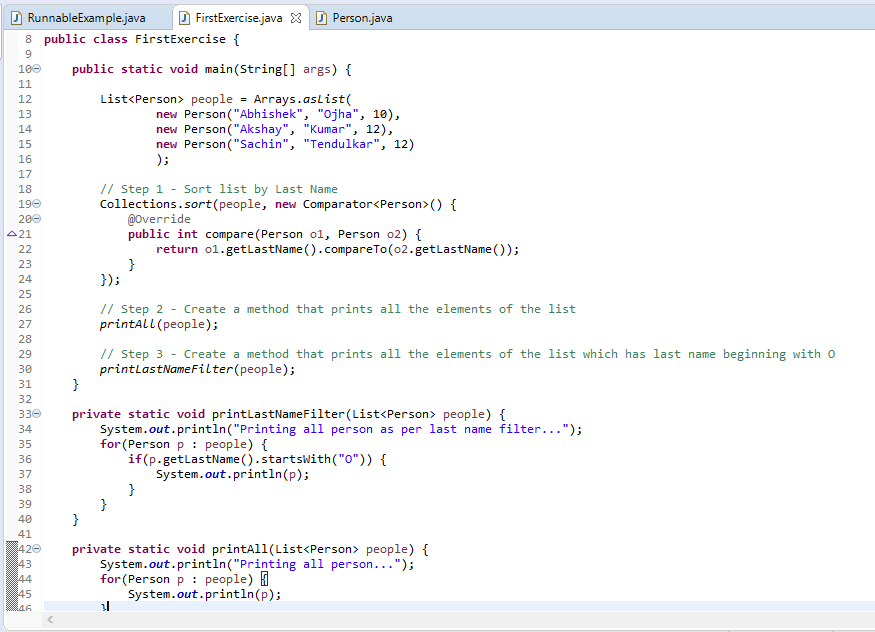
Exercise –

Create a person list and perform different operations both in JAVA 7 way and using Lambda way.

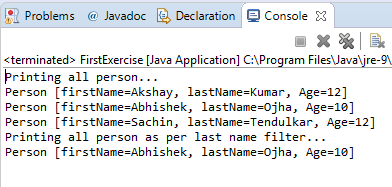
Step1 - Sort list by Last Name

Step2 - Create a method that prints all the elements of the list

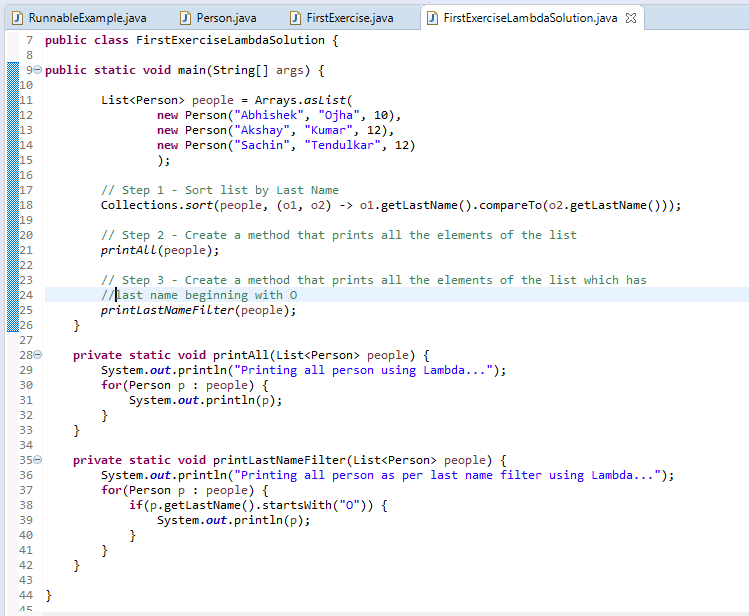
Step3 - Create a method that prints all the elements of the list which has last name beginning with O



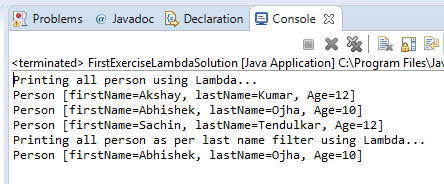
Output –



Above solution using Lambda Expression –



Output –



Functional Interfaces –

There are so many out of the box interfaces are already present in java.util.function package which we can directly use as per our need instead of creating the new one.

Functional interface will have at least one abstract method and other method may define inside interface and for that they have static and default key words.

Ex. Consumer, Predicate and so on

Exception Handling in Lambda Expression –