OOPs with JAVA (BCS403)

Unit: III

Subject Name : Object Oriented Programming with Java (BCS-403)

(B.Tech 4th Sem)

Dr. Birendra Kr. Saraswat

Associate Professor

Department of IT

Java New Features:

Functional Interfaces, Lambda Expression, Method References, Stream API, Default Methods, Static Method, Base64 Encode and Decode, ForEach Method, Try-with resources, Type Annotations, Repeating Annotations, Java Module System, Diamond Syntax with

08

Inner Anonymous Class, Local Variable Type Inference, Switch Expressions, Yield Keyword, Text

Blocks, Records, Sealed Classes

Java Functional Interfaces

An Interface that contains exactly one abstract method is known as functional interface. It can have any number of default, static methods but can contain only one abstract method. It can also declare methods of object class.

Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve functional programming approach.

Java Lambda Expressions

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection. The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code. Java lambda expression is treated as a function, so compiler does not create .class file.

Functional Interface

Lambda expression provides implementation of functional interface. An interface which has only one abstract method is called functional interface. Java provides an anotation @FunctionalInterface, which is used to declare an interface as functional interface.

Why use Lambda Expression

1. To provide the implementation of Functional interface.
2. Less coding.

Java Lambda Expression Syntax

1. (argument-list) -> {body}

Java lambda expression is consisted of three components.

1) Argument-list: It can be empty or non-empty as well.

2) Arrow-token: It is used to link arguments-list and body of expression.

3) Body: It contains expressions and statements for lambda expression.

No Parameter Syntax

```
() -> {
//Body of no parameter lambda
}
```

Java 8 Stream

Java provides a new additional package in Java 8 called java.util.stream. This package consists of classes, interfaces and enum to allows functional-style operations on the elements. You can use stream by importing java.util.streamPackage .

Stream provides following features:

o Stream does not store elements. It simply conveys elements from a source such as a data structure, an array,
or an I/O channel, through a pipeline of computational operations.
o Stream is functional in nature. Operations performed on a stream does not modify it's source. For example,
filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection.
o Stream is lazy and evaluates code only when required.
o The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.

Java Stream Example: Filtering Collection by using Stream

Here, we are filtering data by using stream. You can see that code is optimized and maintained. Stream provides fast execution.

```
import java.util.*;
import java.util.stream.Collectors;
class Product {
int id;
String name;
float price;
public Product(int id, String name, float price) {
this.id = id;
this.name = name;
this.price = price;
} }
public class JavaStreamExample {
public static void main(String[] args) {
List<Product> productsList = new ArrayList<Product>();
//Adding Products
productsList.add(new Product(1,"HP Laptop",25000f));
productsList.add(new Product(2,"Dell Laptop",30000f));
productsList.add(new Product(3,"Lenevo
Laptop",28000f));
productsList.add(new Product(4,"Sony Laptop",28000f));
productsList.add(new Product(5,"Apple
Laptop",90000f));
List<Float> productPriceList2 =productsList.stream()
.filter(p -> p.price > 30000)// filtering data
.map(p->p.price) // fetching price
.collect(Collectors.toList()); // collecting as list
System.out.println(productPriceList2);
}
}
```

Output:

```
[90000.0]
```