Name of the Course : LEARNING JAVA 9–FUNCTIONAL PROGRAMMING

Level : Easy

Tool Stack : Java 9

Problem Statement : If you’ve been using Java 8’s Stream API in conjunction with the Optional class then you might have hit a situation where you want to replace a stream of Optionals with values that are present. ,This is a functional solution, but we can streamline things in Java 9. Here a method on Optional has been added that returns a Stream called, funnily enough, stream(). That will give us a stream with an element in if the Optional has one, or empty otherwise.

Write a program to solve the above task.

Step 1 :  Streams and Parallel Streams

**Code:**

**package com.iiht.java9.functionalprogramming;**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.List;**

**import java.util.stream.Collectors;**

**import java.util.stream.IntStream;**

**public class ComplexFunctional {**

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

**System.out.println("################## Flatmap 1 #####################");**

**flatmap1();**

**System.out.println("################## Flatmap 2 #####################");**

**flatmap2();**

**//Reduce**

**System.out.println("################## Sum Reduce #####################");**

**sumReduce();**

**//Count**

**System.out.println("################## Sum Reduce #####################");**

**count();**

**}**

**public static void count() {**

**long count = IntStream.range(1, 100).count();**

**long sum = IntStream.range(1, 100).sum();**

**System.out.println("Total values between 1 and 100 is " + count);**

**System.out.println("Sum of values between 1 and 100 is " + sum);**

**}**

**private static void sumReduce() {**

**class Person {**

**private final int age;**

**public Person(int age) {**

**this.age = age;**

**}**

**public int getAge() {**

**return this.age;**

**}**

**}**

**List<Person> people = new ArrayList<>();**

**people.add(new Person(10));**

**people.add(new Person(16));**

**people.add(new Person(25));**

**people.add(new Person(65));**

**people.add(new Person(21));**

**people.add(new Person(94));**

**people.add(new Person(25));**

**int totalAge = people.stream()**

**.map(person -> person.getAge())**

**.reduce((a, b) -> a + b)**

**.get();**

**int alsoTotalAge = people.stream().mapToInt(person -> person.getAge()).sum();**

**System.out.println("Total age is " + totalAge);**

**}**

**private static void flatmap1() {**

**List<String> UKCities = Arrays.asList("London", "Manchester", "Birmingham", "Glasgow", "Cardiff");**

**List<String> USCities = Arrays.asList("New York", "Washington DC", "LA", "Miami", "Atlanta");**

**List<String> EUCities = Arrays.asList("Paris", "Brussels", "Munich", "Berlin", "Madrid");**

**List<String> AsianCities = Arrays.asList("Beijing", "Singapore", "Hong Kong", "Jakarta", "Tokyo");**

**List<List<String>> allTheCities = new ArrayList<>();**

**allTheCities.add(UKCities);**

**allTheCities.add(USCities);**

**allTheCities.add(EUCities);**

**allTheCities.add(AsianCities);**

**//I want to combine all the lists**

**allTheCities.stream()**

**.flatMap(cityList -> cityList.stream())**

**.forEach(city -> System.out.println(city));**

**}**

**private static void flatmap2() {**

**LibraryMember member = new LibraryMember();**

**member.addBook("Java 8 in Action");**

**member.addBook("Spring Boot in Action");**

**member.addBook("Effective Java (2nd Edition)");**

**LibraryMember member2 = new LibraryMember();**

**member2.addBook("Learning Python, 5th Edition");**

**member2.addBook("Effective Java (2nd Edition)");**

**List<LibraryMember> list = new ArrayList<>();**

**list.add(member);**

**list.add(member2);**

**List<String> collect =**

**list.stream()**

**.map(x -> x.getBooksOnLoan()) //Stream<Set<String>>**

**.flatMap(x -> x.stream()) //Combines all the streams into one**

**.distinct()**

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

**collect.forEach(x -> System.out.println(x));**

**}**

**}**

Step 2 : Create a class Person in which we will declare a variable called firstName, lastName and use job of class type Job as field to achieve composition relationship

**Code:**

**package** com.iiht.java9.assign2;

**public** **class** Person {

**private** String firstName;

**private** String lastName;

//composition has-a relationship

**private** Job job;

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** Job getJob() {

**return** job;

}

**public** **void** setJob(Job job) {

**this**.job = job;

}

**public** Person(){

**this**.firstName="unknown";

**this**.lastName="unknown";

**this**.job=**new** Job();

job.setSalary(1000L);

}

**public** **long** getSalary() {

**return** job.getSalary();

}

}

Step 3 : Create a main class where we will be creating and object of both Job as well as Person and set Job as a object of Person class and print details of Job as well as Person

**Code:**

**package** com.iiht.java9.assign2;

**public** **class** TestPerson {

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

Person person = **new** Person();

**long** salary = person.getSalary();

Job job=person.getJob();

System.***out***.println("Person Details"+person);

System.***out***.println("Job Details"+job);

System.***out***.println("Salary Details "+salary);

}

}

Learning outcome: Participant could able to learn how to use class composition relationship using Java 9.