Lambda Expression Assignments

1.Write an application to perform basic arithmetic operations like add, subtract, multiply & divide. You need to define a functional interface first

package lambdajava8;

interface Test

{

int operation(int a, int b);

}

public class Arithmetic

{

public static void main(String[] args)

{

Test addition = (int a, int b) -> (a + b);

System.out.println("Addition = " + addition.operation(60, 5));

Test subtraction = (int a, int b) -> (a - b);

System.out.println("Subtraction = " + subtraction.operation(60, 5));

Test multiplication = (int a, int b) -> (a \* b);

System.out.println("Multiplication = " + multiplication.operation(60, 5));

Test division = (int a, int b) -> (a / b);

System.out.println("Division = " + division.operation(60, 5));

}

}

OUTPUT:

Addition = 65

Subtraction = 55

Multiplication = 300

Division = 12

2. Write an application using lambda expressions to print Orders having 2 criteria implemented:

1) order price more than 10000

2) order status is ACCEPTED or COMPLETED.

package lambdajava8;

public interface Price

{

public String price();

}

public class FunctionalInterface

{

public static void main(String[] args) {

Integer i = 200000;

Price p = () -> {

if (i > 10000)

{

return "Accepted!";

} else

return "Not Accepted!";

};

System.out.println(p.price());

}

}

OUTPUT:

Accepted!

3. Use the functional interfaces Supplier, Consumer, Predicate & Function to invoke built-in methods from Java API.

package lambdajava8;

import java.util.function.Consumer;

import java.util.function.Function;

import java.util.function.Predicate;

import java.util.function.Supplier;

class Product

{

private double price = 0.0;

public void setPrice(double price)

{

this.price = price;

}

public void printPrice() {

System.out.println("This is Consumer functional interface: "+price);

}

}

public class third

{

public static void main(String[] args)

{

//consumer functional interface

Consumer<Product>updatePrice = p ->p.setPrice(9.7);

Product p = new Product();

updatePrice.accept(p);

p.printPrice();

//Predicate functional interface

Predicate<String>isALongWord = t ->t.length() > 10;

String s = "successfully";

boolean result = isALongWord.test(s);

System.out.println("This is predicate functional interface: "+s);

//Function functional interface

Function<Integer, Double>half = a ->a / 2.0;

System.out.println("This is Function functional interface: "+half.apply(10));

//Supplier functional interface

Supplier<Double>randomValue = () ->Math.random();

System.out.println("This is supplier functional interface: "+randomValue.get());

}

}

This is Consumer functional interface: 9.7

This is predicate functional interface: successfully

This is Function functional interface: 5.0

This is supplier functional interface: 0.2622511466749796

4. Remove the words that have odd lengths from the list. HINT: Use one of the

new methods from JDK 8. Use removelf() method from Collection interface.

package lambdajava8;

import java.util.\*;

public class Remove

{

public static void main(String[] args)

{

ArrayList<String>str = new ArrayList<String>();

str.add("sunil");

str.add("susheel");

str.add("Ajay");

str.add("Ayush");

str.add("anil");

str.removeIf(w->w.length()%2!=0);

str.forEach(System.out::println);

}

}

OUTPUT:

Ajay

Anil

5. Create a string that consists of the first letter of each word in the list of Strings provided. HINT: Use Consumer interface & a StringBuilder to construct the result.

package lambdajava8;

import java.util.Arrays;

import java.util.List;

import java.util.function.Consumer;

public class StringBuilder

{

public static void main(String a[])

{

List<String>strings = Arrays.asList("Ram", "Shyam", "Ajay", "Susheel", "Anil");

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

strings.stream()

.map(s ->s.substring(0, 1))

.forEach(s ->System.out.println(s));

}

}

OUTPUT:

R

S

A

S

A

6.Replace every word in the list with its upper case equivalent. Use replaceAll() method &UnaryOperator interface.

package lambdajava8;

import java.util.ArrayList;

import java.util.function.UnaryOperator;

class Op implements UnaryOperator<String>{

public String apply(String str)

{

return str.toUpperCase();

}

}

public class ReplaceMethod

{

public static void main(String[] args)

{

ArrayList<String>list = new ArrayList<>();

list.add("java");

list.add("is ");

list.add("my");

list.add("favourite");

list.add("subject");

System.out.println("Contents of the list: "+list);

list.replaceAll(new Op());

System.out.println("Contents of the list after replace operation: \n"+list);

}

}

OUTPOT:

Contents of the list: [java, is , my, favourite, subject]

Contents of the list after replace operation:

[JAVA, IS , MY, FAVOURITE, SUBJECT]

7. Convert every key-value pair of the map into a string and append them all into a single string, in iteration order. HINT: Use Map.entrySet() method & a StringBuilder to construct the result String.

package lambdajava8;

import java.util.HashMap;

import java.util.Map;

import java.util.stream.Collectors;

public class MaptoString {

public static void main(String[] cmd\_lineParams) {

Map<String, String> map = new HashMap<>(5);

map.put("Susheel", "1");

map.put("Ram", "2");

map.put("anil", "3");

map.put("Rajesh", "6");

map.put("ameesha", "5");

String s = map.entrySet().stream().map((entry) ->

"" + entry.getKey() + " \"" + entry.getValue().replaceAll("\"", "\\\\\"") + "\"")

.collect(Collectors.joining(" "));

System.out.println(s);

}

}

OUTPUT:

Rajesh "6" ameesha "5" Susheel "1" anil "3" Ram "2"

8. Create new thread that prints the numbers from the list. Use class Thread & Interface Consumer.

package lambdajava8;

import java.util.ArrayList;

import java.util.List;

public class ThreadList {

public static void main(String[] args) {

// TODO Auto-generated method stub

List<Integer> num=new ArrayList<Integer>(){{

add(11);

add(55);

add(37);

add(95);

add(99);

}};

Thread mylambda = new Thread(()->System.out.println(num));

mylambda.run();

}

}

OUTPUT:

[11, 55, 37, 95, 99]