Lambda expression assignments

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

@functional Interface

interface Arithmetic {

int operations(int a, int b);

}

public class LambdaExample1 {

public static void main (String[] args) {

//addition operation

Arithmetic addition = (a, b) -> (a + b);

System.out.println(“Addition -> “+ addition.operations (20,10));

//subtraction operation

Arithmetic subtraction = (a, b) -> (a - b);

System.out.println(“Subtraction -> “+ subtraction.operations (20,10));

//division operation

Arithmetic division = (a, b) -> (a / b);

System.out.println(“Division -> “+ division.operations (20,10));

//multiplication operation

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

System.out.println(“Multiplication -> “+ multiplication.operations (20,10));

}

}

Output

Addition -> 30

Subtraction -> 10

Division -> 2

Multiplication -> 200

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.

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

import java.util.function.\*;

import java.util.stream.Stream;

class order {

int orderId;

Double cost;

String orderStatus;

public order(int orderId, Double cost, String orderStatus) {

super();

this.orderId = orderId;

this.cost = cost;

this.orderStatus = orderStatus;

}

public int getOrderId() {

return orderId;

}

public void setOrderId(int orderId) {

this.orderId = orderId;

}

public Double getCost() {

return cost;

}

public void setCost(Double cost) {

this.cost = cost;

}

public String getOrderStatus() {

return orderStatus;

}

public void setOrderStatus(String orderStatus) {

this.orderStatus = orderStatus;

}

@Override

public String.toString() {

// TODO Auto-generated method stub

//return super.toString();

return "order [orderId=" + orderId + ", cost=" + cost + ", orderStatus=" + orderStatus + "]";

}

}

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Stream;

public class Price {

public static void main(String[] args) {

// TODO Auto-generated method stub

List<order> list=new ArrayList<order>();

list.add(new order(1,2500.0,"Accepted"));

list.add(new order(2,10000.0,"Completed"));

list.add(new order(3,15000.0,"Accepted"));

list.add(new order(4,9500.0,"Completed"));

//printing all the order price more than 10000 and printing its status

Stream<order> filtered\_data = list.stream().filter(o -> o.cost >= 10000);

filtered\_data.forEach(  order -> System.out.println(order.cost+": "+order.orderStatus));

}

}

Output

10000.0: Completed

15000.0: Accepted

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

import java.util.function.Consumer;

import java.util.function.Function;

import java.util.function.Predicate;

import java.util.function.Supplier;

public class interfaces {

public static void main(String[] args) {

//consumer functional interface

String str = “Consumer Interface”;

Consumer<String>displayConsumer = a -> System.out.println(a);

displayConsumer.accept(str.to.UpperCase());

//Predicate functional interface

Predicate<String>displaypredicate = p ->str.length() > 10;

System.out.println("Predicate functional interface: "+displaypredicate.test(str));

//Function functional interface

Function<Integer, Double>val = a ->a / 5.0;

System.out.println("Function functional interface: "+val.apply(37));

//Supplier functional interface

Supplier<Float>suppval = () ->Math.max(18.99f,19.9f);

System.out.println("Supplier functional interface: "+suppval.get());

}

}

Output Consumer Interface

Predicate functional interface: true

Function functional interface: 7.4

Supplier functional interface: 19.9

4. Remove the words that have odd lengths from the list. HINT: Use one of the new methods from JDK 8.Use removeif() method from Collection interface.

import java.util.\*;

public class words {

public static void main(String[] args) {

  // TODO Auto-generated method stub

   List<String> word = new ArrayList<String>();

         word.add("Hello");

         word.add("World");

         word.add("Laptop");

         word.add("Window");

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

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

  }

}

Output

Laptop

Window

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

import java.util.Arrays;

import java.util.List;

import java.util.function.Consumer;

public class AppendResult {

public static void main(String[] args) {

// TODO Auto-generated method stub

StringBuilder str=new StringBuilder();

List<String> names =Arrays.asList(

new String("What"), new String("Else"),

new String("Look"), new String("Like"),

new String("Dust"), new String("Or"),

new String("Not"),new String("Edible"));

for(String n : names) {

str.append(n.charAt(0));

}

//prints the first letters of all the string in the list

printstring(str,c->System.out.println(str));

}

private static void printstring(StringBuilder str,Consumer consumer) {

// TODO Auto-generated method stub

if(str!=null) {

consumer.accept(str);

}

}

}

Output:

WELLDONE

6. Replace every word in the list with its upper case equivalent. Use replaceAll method & Unary Operator interface.

import java.util.Arrays;

import java.util.List;

import java.util.function.UnaryOperator;

class replace implements UnaryOperator<String>{

public String apply(String str) {

      return str.toUpperCase();

    }

}

public class Unaryopt {

 public static void main(String[] args) {

  // TODO Auto-generated method stub

  List<String> names =Arrays.asList(

    new String("What"), new String("Else"),

    new String("Look"), new String("Like"),

    new String("Dust"), new String("Or"),

    new String("Not"),new String("Edible")

    );

  System.out.println("list before replace operation: "+names);

  names.replaceAll(new replace());

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

  }

}

Output

list before replace operation: [What, Else, Look, Like, Dust, Or, Not, Edible]

Contents of the list after replace operation:

[WHAT, ELSE, LOOK, LIKE, DUST, OR, NOT, EDIBLE]

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 String Builder to construct the result String.

import java.util.HashMap;

import java.util.Map;

import java.util.stream.Collectors;

public class maptostring {

public static void main(String[] args) {

// TODO Auto-generated method stub

Map<String, String> map = new HashMap<>(4);

map.put("Hello", "World");

map.put("String", "Buffer");

map.put("String", "Builder");

map.put(“Data”, “Model”);

String s = map.entrySet().stream().map((entry) ->" " + entry.getKey() + " " + entry.getValue().replaceAll("\s ", "\\s ") + " ").collect(Collectors.joining(" "));

System.out.println(s);

}

}

Output

Hello World

String Buffer

String Builder

Data Model

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

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(22);

add(33);

add(44);

add(55);

}

};

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

mylambda.run();

}

}

Output

[11, 22, 33, 44, 55]