1.Write the following a functional interface and implement it using lambda:

a)First number is greater than second number or not

interface MyInterface{

boolean perform(int a, int b);

}

public class FunctionalInterfaceLambdaEx {

public static void main(String[] args) {

MyInterface mi = (int a, int b)->{

if (a>b)

return true;

else

return false;

};

boolean res = mi.perform(7,9);

System.out.println(res);

}

}

b)Increment the number by 1 and return incremented value Parameter (int) Return int

interface MyInterface{

int perform(int a);

}

public class FunctionalInterfaceLambdaEx {

public static void main(String[] args) {

MyInterface mi = (int a)->{

a=++a;

return a;

};

int res = mi.perform(7);

System.out.println(res);

}

}

c)Concatination of 2 string Parameter (String , String ) Return (String)

interface MyInterface{

String perform(String s1, String s2);

}

public class FunctionalInterfaceLambdaEx {

public static void main(String[] args) {

MyInterface mi = (String s, String s1)->{

String s2 = s + s1;

return s2;

};

String res = mi.perform("Shobhit","Bisaria");

System.out.println(res);

}

}

d)Convert a string to uppercase and return . Parameter (String) Return (String)

interface MyInterface{

String perform(String s);

}

public class FunctionalInterfaceLambdaEx {

public static void main(String[] args) {

MyInterface mi = (String s)->{

String s1 = s.toUpperCase();

return s1;

};

String res = mi.perform("Shobhit");

System.out.println(res);

}

}

2.

interface Functional{

int print(int a, int b);

}

public class FunctionalEx {

public static void main(String[] args) {

Functional fun=(int x, int y)->{

return (x+y);

};

int val = fun.print(5,7);

System.out.println(val);

}

}

3.

a)add

interface Reference{

void display();

}

public class MethodRefernceEx {

void add(){

int a=7;

int b=3;

int c=a+b;

System.out.println(c);

}

public static void main(String[] args) {

Reference ref = new MethodRefernceEx()::add;

ref.display();

}

}

b)subtract

interface Reference{

void display();

}

public class MethodRefernceEx {

void add(){

int a=7;

int b=3;

int c=a-b;

System.out.println(c);

}

public static void main(String[] args) {

Reference ref = new MethodRefernceEx()::add;

ref.display();

}

}

c)

interface MethodRefernce {

void disp();

}

public class StaticMethodReferenceEx {

static void multiply() {

int a = 5;

int b = 5;

int c = a \* b;

System.out.println(c);

}

public static void main(String[] args) {

MethodRefernce ref = StaticMethodReferenceEx::multiply;

ref.disp();

}

}

4.

//Employee class

public class Employee {

String name;

int age;

String city;

public Employee(String s, int i, String c) {

System.out.println(s+" "+i+" "+c);

}

}

//Driver code

interface EmployeeRef{

public abstract Employee getEmpDetails(String n, int a, String c);

}

public class EmployeeReference {

public static void main(String[] args) {

EmployeeRef er = Employee::new;

er.getEmpDetails("Shobhit",24,"Kanpur");

}

}

5.

a)Consumer

import java.util.function.Consumer;

public class ConsumerInterface {

public static void main(String[] args) {

Consumer consumer = (i)->{

System.out.println(i);

};

consumer.accept(5);

}

}

b)Supplier

import java.util.function.Supplier;

public class SupplierInterface {

public static void main(String[] args) {

Supplier supplier = ()->{

return 5;

};

System.out.println(supplier.get());

}

}

c)Predicate

import java.util.function.Predicate;

public class PredicateInterface {

public static void main(String[] args) {

Predicate<Integer> predicate = (e)->{

return e>2;

};

System.out.println(predicate.test(6));

}

}

d)Function

import java.util.function.Function;

public class FunctionInterface {

public static void main(String[] args) {

Function<Integer,Integer> function = (e)->{

return e\*2;

};

System.out.println(function.apply(5));

}

}

6.

interface DefaultInterface{

default void newMethod(){

System.out.println("default in interface");

}

void normal();

}

public class InterfaceDefaultEx implements DefaultInterface{

public static void main(String[] args) {

InterfaceDefaultEx obj = new InterfaceDefaultEx();

obj.normal();

obj.newMethod();

}

@Override

public void normal() {

System.out.println("abstract in interface");

}

}

interface StaticMethodInterface{

default void first(){

System.out.println("default");

}

static void staticMethod(){

System.out.println("static method in interface");

}

}

public class StaticMethodEx implements StaticMethodInterface {

public static void main(String[] args) {

StaticMethodEx s = new StaticMethodEx();

s.first();

StaticMethodInterface.staticMethod();

}

}

7.

interface One{

default void print(){

System.out.println("In one");

}

void norm();

}

interface Two{

default void print(){

System.out.println("In two");

}

void nor();

}

public class DefaultOverrideEx implements One, Two{

public static void main(String[] args) {

DefaultOverrideEx def = new DefaultOverrideEx();

def.print();

}

@Override

public void nor() {

}

@Override

public void norm() {

}

@Override

public void print(){

System.out.println("In class");

}

}

8.

interface One{

default void print(){

System.out.println("In one");

}

void norm();

}

interface Two{

default void print(){

System.out.println("In two");

}

void nor();

}

public class DefaultOverrideEx implements One, Two{

public static void main(String[] args) {

DefaultOverrideEx def = new DefaultOverrideEx();

def.print();

}

@Override

public void nor() {

}

@Override

public void norm() {

}

@Override

public void print(){

One.super.print();

Two.super.print();

}

}

9.

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class CollectEx {

public static void main(String[] args) {

List<Integer> list = Arrays.asList(1,2,3,4,5,5,6,7,8,9,0);

System.out.println(

list

.stream()

.filter(e->e%2==0)

.collect(Collectors.toList())

);

}

}

10.

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class CollectSumEx {

public static void main(String[] args) {

List<Integer> list = Arrays.asList(1,2,3,4,5,6,7);

System.out.println(

list.stream().filter(e->e>5)

.collect(Collectors.summingInt(e->e))

);

}

}

11.

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class CollectAverageEx {

public static void main(String[] args) {

List<Integer> list = Arrays.asList(1,2,3,4,5);

System.out.println(

list.stream()

.collect(Collectors.averagingInt(e->e\*2))

);

}

}

12.

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FirstEvenEx {

public static void main(String[] args) {

List<Integer> list = Arrays.asList(1,2,3,4,5,6,7,8,9,10);

System.out.println(

list

.stream()

.filter(e->e>3)

.filter(e->e%2==0)

.findFirst()

);

}

}