* **Write the following a functional interface and implement it using lambda:**
  + **(1) First number is greater than second number or not**
  + **Parameter (int ,int ) Return boolean**

package ques1;

import java.util.Scanner;

public class Ques1\_1 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter two numbers");

int firstNum = sc.nextInt();

int secondNum = sc.nextInt();

greaterNumber fs = (Num1,Num2) -> Num1>Num2;

boolean result = fs.isFirstNumberGreaterThanSecond(firstNum,secondNum);

System.out.println(result);

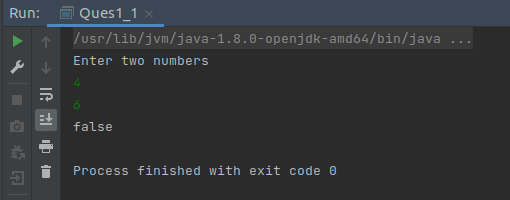
}

}

interface greaterNumber{

boolean isFirstNumberGreaterThanSecond(int first,int second);

}



* + **(2) Increment the number by 1 and return incremented value**
  + **Parameter (int) Return int**

package ques1;

import java.util.Scanner;

public class Ques1\_2 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a number");

int n = sc.nextInt();

incrementNumber in = (num) -> num + 1;

System.out.println("incremented number " + in.incrementNum(n));

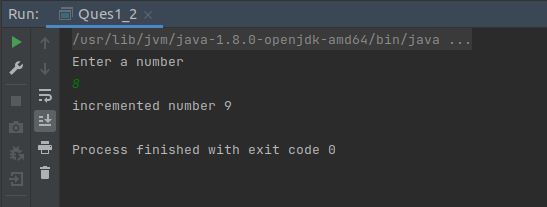
}

}

interface incrementNumber{

int incrementNum(int num);

}



* + **(3) Concatination of 2 string**
  + **Parameter (String , String ) Return (String)**

package ques1;

import java.util.Scanner;

public class Ques1\_3 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter two strings");

String str1 = sc.next();

String str2 = sc.next();

Concat concatStrings = (s1,s2) -> s1 + s2;

System.out.println(concatStrings.concat2Strings(str1,str2));

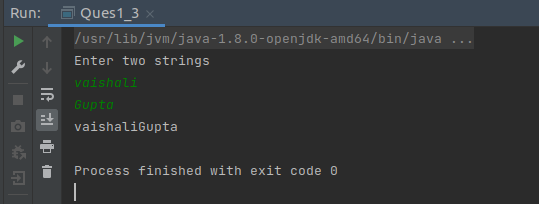
}

}

interface Concat{

String concat2Strings(String str1,String str2);

}



* + **(4) Convert a string to uppercase and return .**
  + **Parameter (String) Return (String)**

package ques1;

import java.util.Locale;

import java.util.Scanner;

public class Ques1\_4 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a String");

String str = sc.next();

uppercase up = s -> s.toUpperCase();

System.out.println(up.convertIntoUppercase(str));

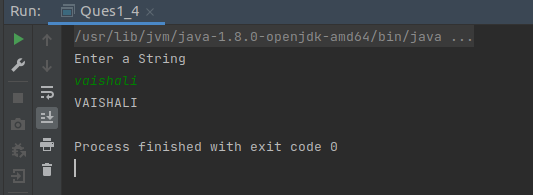
}

}

interface uppercase{

String convertIntoUppercase(String str);

}



* **Create a functional interface whose method takes 2 integers and return one integer.**

import java.util.Scanner;

public class Ques2 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter two numbers");

int num1 = sc.nextInt();

int num2 = sc.nextInt();

Operation op = (n1,n2) -> n1+n2;

System.out.println("Sum of numbers = " + op.sum2nums(num1,num2));

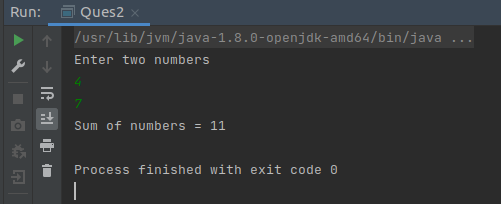
}

}

interface Operation{

int sum2nums(int num1,int num2);

}



* **Using (instance) Method reference create and apply add and subtract method and using (Static) Method reference create and apply multiplication method for the functional interface created.**

import java.util.Scanner;

public class Ques3 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Please Enter Two Numbers");

int num1 = sc.nextInt();

int num2 = sc.nextInt();

Calculator calc = new MyCalculator()::sum;

System.out.println("sum = " + calc.calculate(num1,num2));

Calculator calc1 = new MyCalculator()::subtract;

System.out.println("Difference = " + calc1.calculate(num1,num2));

Calculator calc2 = MyCalculator::multiply;

System.out.println("Product = " + calc2.calculate(num1,num2));

}

}

@FunctionalInterface

interface Calculator {

int calculate(int num1, int num2);

}

class MyCalculator {

public int sum(int num1, int num2 ) {

return num1 + num2;

}

public int subtract(int num1, int num2 ) {

return num1 > num2 ? num1 - num2 : num2 - num1;

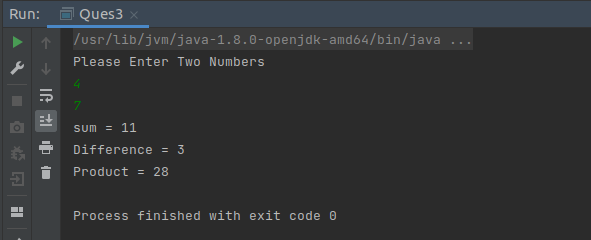
}

public static int multiply(int num1, int num2 ) {

return num1 \* num2;

}

}



* **Create an Employee Class with instance variables (String) name, (Integer)age, (String)city and get the instance of the Class using constructor reference**

public class Ques4 {

public static void main(String[] args) {

EmployeeInstances employee = Employee::new;

Employee emp= employee.getEmployee("vaishali", 24,"Faridabad");

System.out.println(emp);

}

}

class Employee{

String name;

int age;

String city;

Employee(){}

Employee(String name, int age, String city){

this.name = name;

this.age = age;

this.city = city;

}

@Override

public String toString() {

return "Employee{" +

"name='" + name + '\'' +

", age=" + age +

", city='" + city + '\'' +

'}';

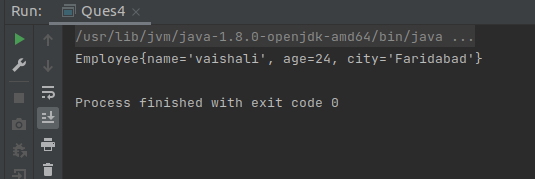
}

}

interface EmployeeInstances{

Employee getEmployee(String name,int age,String city);

}



* **Implement following functional interfaces from java.util.function using lambdas:**
  + **(1) Consumer**
  + **(2) Supplier**
  + **(3) Predicate**
  + **(4) Function**

import java.util.function.Consumer;

import java.util.function.Function;

import java.util.function.Predicate;

import java.util.function.Supplier;

public class Ques5 {

public static void main(String[] args) {

//A supplier does the opposite of a consumer,

// it takes no arguments and only returns some value.

System.out.println("Supplier Functional Interface");

String t = "One";

Supplier<String> supplierStr = () -> t.toUpperCase();

System.out.println("Uppercase String = " + supplierStr.get());

// The Function interface represents a function (method) that

// takes a single parameter and returns a single value.

System.out.println("Function Interface");

Function<Long, Long> adder = (value) -> value + 3;

Long resultLambda = adder.apply((long) 8);

System.out.println("result = " + resultLambda);

//a Predicate is a functional interface that

// can be used anywhere you need to evaluate a boolean condition.

System.out.println("Predicate Interface");

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

String s = "successfully";

boolean result = isALongWord.test(s);

System.out.println("String length greater than 10 " + result);

//A consumer is an operation that

// accepts a single input argument and returns no result;

// it just execute some operations on the argument.

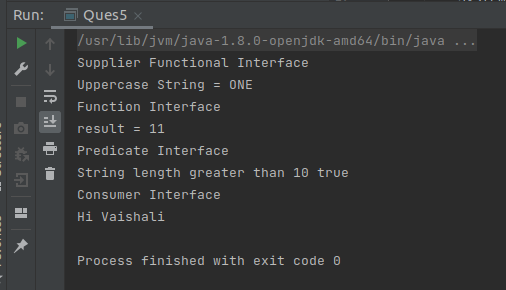
System.out.println("Consumer Interface");

Consumer<String> consumeStr = str -> System.out.println("Hi " + str);

consumeStr.accept("Vaishali");

}

}



* **Create and access default and static method of an interface.**

public class Ques6 {

public static void main(String[] args) {

Implementor obj = new Implementor();

//calling the default method of interface

obj.newDefaultMethod();

//calling the static method of interface

MyInterface.newStaticMethod();

//calling the abstract method of interface

obj.existingMethod("existing abstract method");

}

}

interface MyInterface{

//default method

default void newDefaultMethod(){

System.out.println("Newly added default method");

}

//static method

static void newStaticMethod(){

System.out.println("Newly added static method");

}

/\* Already existing public and abstract method

\* We must need to implement this method in

\* implementation classes.

\*/

void existingMethod(String str);

}

class Implementor implements MyInterface{

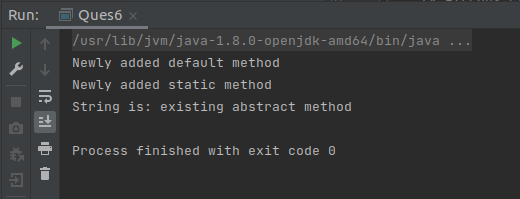
// implementing abstract method

public void existingMethod(String str){

System.out.println("String is: "+str);

}

}



* **Override the default method of the interface.**

public class Ques7 {

public static void main(String[] args) {

ImplementorClass obj = new ImplementorClass();

//calling the default method of interface

obj.newDefaultMethod();

}

}

interface MyInterface1{

//default method

default void newDefaultMethod(){

System.out.println("Newly added default method");

}

}

class ImplementorClass implements MyInterface1{

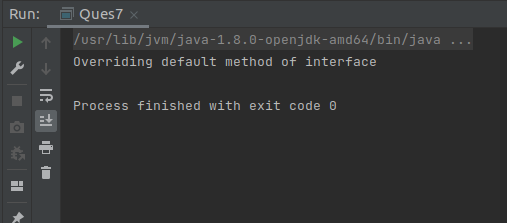
//overriding default method

public void newDefaultMethod(){

System.out.println("Overriding default method of interface");

}

}



* **Implement multiple inheritance with default method inside interface.**

public class Ques8 {

public static void main(String[] args) {

implementor1 obj = new implementor1();

obj.display();

}

}

interface MyInterface\_1{

public static int num = 100;

public default void display() {

System.out.println("display method of MyInterface1");

}

}

interface MyInterface\_2{

public static int num = 1000;

public default void display() {

System.out.println("display method of MyInterface2");

}

}

class implementor1 implements MyInterface\_1, MyInterface\_2{

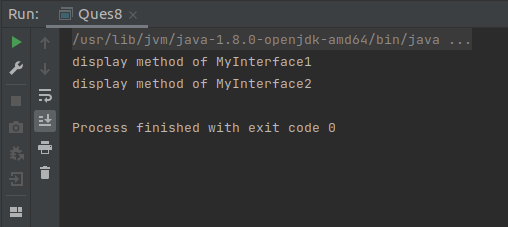
public void display() {

MyInterface\_1.super.display();

MyInterface\_2.super.display();

}

}



* **Collect all the even numbers from an integer list.**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class Ques9 {

static List<Integer> collectEven(List<Integer> list){

return list.stream().filter(x -> x%2 == 0 ).collect(Collectors.toList());

}

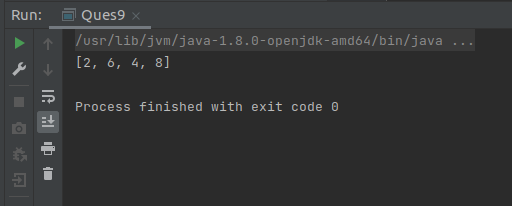
public static void main(String[] args) {

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

System.out.println(collectEven(list));

}

}



* **Sum all the numbers greater than 5 in the integer list.**

import java.util.Arrays;

import java.util.List;

public class Ques10 {

public static void main(String[] args) {

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

System.out.println("Sum of numbers greater than 5 = " +

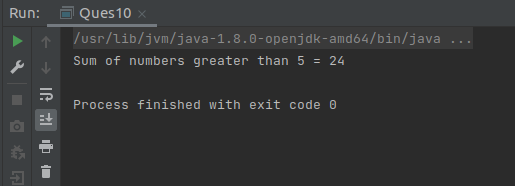
list.stream().filter(x -> x>5)

.mapToInt(x ->x)

.sum());

}

}



* **Find average of the number inside integer list after doubling it.**

import java.util.Arrays;

import java.util.List;

import java.util.OptionalDouble;

public class Ques11 {

public static void main(String[] args) {

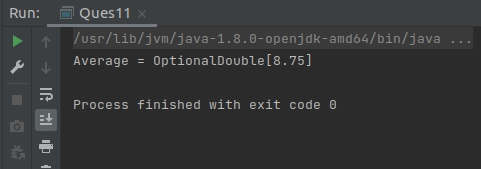
List<Integer> list = Arrays.asList(2,1,4,6,3,7,8,4);

OptionalDouble avg = list.stream().mapToInt(x -> 2\*x).average();

System.out.println("Average = " + avg);

}

}



* **Find the first even number in the integer list which is greater than 3.**

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

public class Ques12 {

public static void main(String[] args) {

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

Optional<Integer> firstEvenNum = list.stream()

.filter(x -> x>3)

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

.findFirst();

System.out.println("First Even Number = " + firstEvenNum);

}

}

