1. **Answer**

public class Student {

private String name;

private int rollNumber;

private int marks;

public Student(String name, int rollNumber, int marks) {

this.name = name;

this.rollNumber = rollNumber;

if (marks >= 0 && marks <= 100) {

this.marks = marks;

} else {

this.marks = 0;

}

}

public String getName() {

return name;

}

public int getRollNumber() {

return rollNumber;

}

public int getMarks() {

return marks;

}

public void displayDetails() {

System.***out***.println("name: " + name);

System.***out***.println("rolno: " + rollNumber);

System.***out***.println("marks: " + marks);

}

public static void main(String[] args) {

Student student1 = new Student("Anush", 101, 85);

student1.displayDetails();

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

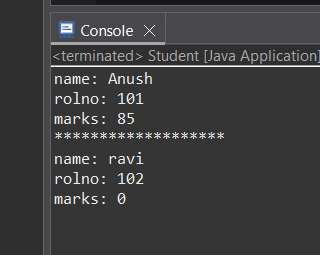
Student student2 = new Student("ravi", 102, 150);

student2.displayDetails();

}

}

**OUTPUT:**



1. **Answer**

public class Rectangle {

private double width;

private double height;

public Rectangle(double width, double height) {

if (width <= 0 || height <= 0) {

throw new IllegalArgumentException("width and height must positive");

}

this.width = width;

this.height = height;

}

public void setWidth(double width) {

if (width <= 0) {

throw new IllegalArgumentException("width must positive.");

}

this.width = width;

}

public void setHeight(double height) {

if (height <= 0) {

throw new IllegalArgumentException("height must positive.");

}

this.height = height;

}

public double getWidth() {

return width;

}

public double getHeight() {

return height;

}

public double getArea() {

return width \* height;

}

public double getPerimeter() {

return 2 \* (width + height);

}

public void displayDetails() {

System.out.println("rectangle:");

System.out.println("width: " + width);

System.out.println("height: " + height);

System.out.println("area: " + getArea());

System.out.println("perimeter: " + getPerimeter());

}

public static void main(String[] args) {

try {

Rectangle rect = new Rectangle(5, 10);

rect.displayDetails();

rect.setWidth(7);

rect.setHeight(3);

rect.displayDetails();

} catch (Exception e) {

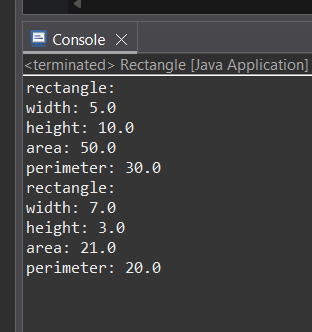
System.out.println("error: " + e.getMessage());

}

}

}

**OUTPUT:**

****

1. **Answer**

import java.util.ArrayList;

import java.util.List;

public class BankAccount {

private String accountNumber;

private String accountHolder;

private double balance;

private List<Transaction> transactions;

private class Transaction {

private String type;

private double amount;

private double postBalance;

public Transaction(String type, double amount, double postBalance) {

this.type = type;

this.amount = amount;

this.postBalance = postBalance;

}

*@Override*

public String toString() {

return String.*format*("%s: %.2f, Balance after: %.2f", type, amount, postBalance);

}

}

public BankAccount(String accountNumber, String accountHolder, double initialBalance) {

if (initialBalance < 0) {

throw new IllegalArgumentException("Initial balance cannot be negative.");

}

this.accountNumber = accountNumber;

this.accountHolder = accountHolder;

this.balance = initialBalance;

this.transactions = new ArrayList<>();

}

public void deposit(double amount) {

if (amount <= 0) {

System.***out***.println("Deposit amount must be positive.");

return;

}

balance += amount;

transactions.add(new Transaction("Deposit", amount, balance));

}

public boolean withdraw(double amount) {

if (amount <= 0) {

System.***out***.println("withdraw amount must be positive.");

return false;

}

if (amount > balance) {

System.***out***.println("insufficient balance for withdrawal.");

return false;

}

balance -= amount;

transactions.add(new Transaction("Withdraw", amount, balance));

return true;

}

public double getBalance() {

return balance;

}

public String getLastTransaction() {

if (transactions.isEmpty()) {

return "no transactions available.";

}

return transactions.get(transactions.size() - 1).toString();

}

*@Override*

public String toString() {

String maskedAccNum = "\*\*\*\*" + accountNumber.substring(Math.*max*(0, accountNumber.length() - 4));

return String.*format*("account Holder: %s\naccount Number: %s\nbalance: %.2f",

accountHolder, maskedAccNum, balance);

}

public static void main(String[] args) {

BankAccount acc = new BankAccount("1234567890123456", "Anush Rao", 1000);

System.***out***.println(acc);

acc.deposit(500);

System.***out***.println("after deposit: " + acc.getBalance());

System.***out***.println("last transaction: " + acc.getLastTransaction());

boolean success = acc.withdraw(2000);

System.***out***.println("withdraw 2000 successful? " + success);

System.***out***.println("balance: " + acc.getBalance());

success = acc.withdraw(300);

System.***out***.println("withdraw 300 successful? " + success);

System.***out***.println("balance: " + acc.getBalance());

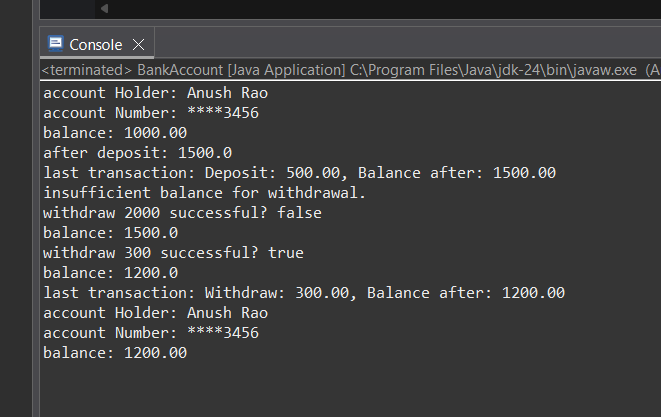
System.***out***.println("last transaction: " + acc.getLastTransaction());

System.***out***.println(acc);

}

}

**OUTPUT:**



**4.Answer**

public class Locker {

private String lockerId;

private boolean isLocked;

private String passcode;

private SecurityManager securityManager;

private class SecurityManager {

private boolean verifyPasscode(String code) {

return passcode.equals(code);

}

}

public Locker(String lockerId, String passcode) {

this.lockerId = lockerId;

this.passcode = passcode;

this.isLocked = true;

this.securityManager = new SecurityManager();

}

public void lock() {

isLocked = true;

System.***out***.println("Locker " + lockerId + " is now locked.");

}

public boolean unlock(String code) {

if (securityManager.verifyPasscode(code)) {

isLocked = false;

System.***out***.println("Locker " + lockerId + " unlocked successfull.");

return true;

} else {

System.***out***.println("incorrect passcode. not unlocked.");

return false;

}

}

public boolean isLocked() {

return isLocked;

}

public static void main(String[] args) {

Locker myLocker = new Locker("L001", "secret123");

System.***out***.println("initially locked? " + myLocker.isLocked());

boolean success = myLocker.unlock("wrongpass");

System.***out***.println("unlock attempt with wrong code: " + success);

System.***out***.println("is locked? " + myLocker.isLocked());

success = myLocker.unlock("secret123");

System.***out***.println("unlock attempt with correct code: " + success);

System.***out***.println("is locked? " + myLocker.isLocked());

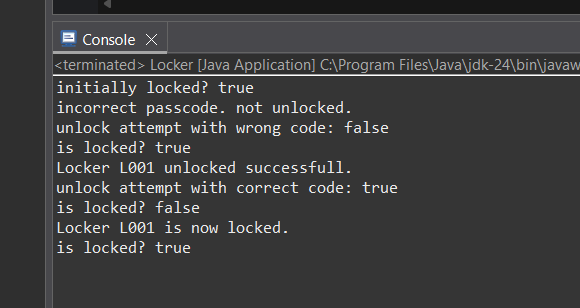
myLocker.lock();

System.***out***.println("is locked? " + myLocker.isLocked());

}

}

**OUTPUT:**



1. **Answer**

public final class Product {

private final String name;

private final String code;

private final double price;

private final String category;

private Product(Builder builder) {

this.name = builder.name;

this.code = builder.code;

this.price = builder.price;

this.category = builder.category;

}

public String getName() {

return name;

}

public String getCode() {

return code;

}

public double getPrice() {

return price;

}

public String getCategory() {

return category;

}

public static class Builder {

private String name;

private String code;

private double price;

private String category;

public Builder withName(String name) {

if (name == null || name.isEmpty()) {

throw new IllegalArgumentException("name cannot be empty");

}

this.name = name;

return this;

}

public Builder withCode(String code) {

if (code == null || code.isEmpty()) {

throw new IllegalArgumentException("code cannot be empty");

}

this.code = code;

return this;

}

public Builder withPrice(double price) {

if (price < 0) {

throw new IllegalArgumentException("price cannot be negative");

}

this.price = price;

return this;

}

public Builder withCategory(String category) {

if (category != null && category.isEmpty()) {

throw new IllegalArgumentException("category cannot be empty");

}

this.category = category;

return this;

}

public Product build() {

if (this.name == null) {

throw new IllegalStateException("name required.");

}

if (this.code == null) {

throw new IllegalStateException("code needed");

}

return new Product(this);

}

}

public static void main(String[] args) {

try {

Product product = new Product.Builder()

.withName("laptop")

.withCode("lp1001")

.withPrice(1500.00)

.withCategory("electronics")

.build();

System.***out***.println("product created");

System.***out***.println("name: " + product.getName());

System.***out***.println("code: " + product.getCode());

System.***out***.println("price: " + product.getPrice());

System.***out***.println("categery: " + product.getCategory());

} catch (Exception e) {

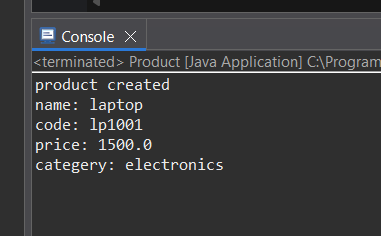
System.***out***.println("Error: " + e.getMessage());

}

}

}

**OUTPUT:**

****

**Interface:**

1. **Answer**

public class BkwSeq implements CharSequence {

private final String reversed;

public BkwSeq(String input) {

if (input == null) {

throw new IllegalArgumentException("Input cannot be empty");

}

this.reversed = new StringBuilder(input).reverse().toString();

}

@Override

public int length() {

return reversed.length();

}

@Override

public char charAt(int index) {

if (index < 0 || index >= length()) {

throw new IndexOutOfBoundsException("Index out of range");

}

return reversed.charAt(index);

}

@Override

public CharSequence subSequence(int start, int end) {

if (start < 0 || end > length() || start > end) {

throw new IndexOutOfBoundsException("Invalid subseq");

}

return reversed.substring(start, end);

}

@Override

public String toString() {

return reversed;

}

public static void main(String[] args) {

BkwSeq seq = new BkwSeq("hello");

System.out.println("string: " + seq.toString());

System.out.println("len: " + seq.length());

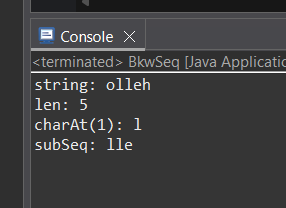
System.out.println("charAt(1): " + seq.charAt(1));

System.out.println("subSeq: " + seq.subSequence(1, 4));

}

}

**OUTPUT:**

****

1. **Answer**

interface Movable {

void moveUp();

void moveDown();

void moveLeft();

void moveRight();

}

class MovablePoint implements Movable {

int x, y;

int xSpeed, ySpeed;

public MovablePoint(int x, int y, int xSpeed, int ySpeed) {

this.x = x;

this.y = y;

this.xSpeed = xSpeed;

this.ySpeed = ySpeed;

}

@Override

public void moveUp() {

y += ySpeed;

}

@Override

public void moveDown() {

y -= ySpeed;

}

@Override

public void moveLeft() {

x -= xSpeed;

}

@Override

public void moveRight() {

x += xSpeed;

}

@Override

public String toString() {

return "Point(" + x + ", " + y + "), Speed(" + xSpeed + ", " + ySpeed + ")";

}

}

class MovableCircle implements Movable {

int radius;

MovablePoint center;

public MovableCircle(int radius, MovablePoint center) {

this.radius = radius;

this.center = center;

}

@Override

public void moveUp() {

center.moveUp();

}

@Override

public void moveDown() {

center.moveDown();

}

@Override

public void moveLeft() {

center.moveLeft();

}

@Override

public void moveRight() {

center.moveRight();

}

@Override

public String toString() {

return "Circle[Radius=" + radius + ", Center=" + center + "]";

}

}

class MovableRectangle implements Movable {

MovablePoint topLeft;

MovablePoint bottomRight;

public MovableRectangle(MovablePoint topLeft, MovablePoint bottomRight) {

if (topLeft.xSpeed != bottomRight.xSpeed || topLeft.ySpeed != bottomRight.ySpeed) {

throw new IllegalArgumentException("Both points must have the same speed");

}

this.topLeft = topLeft;

this.bottomRight = bottomRight;

}

@Override

public void moveUp() {

topLeft.moveUp();

bottomRight.moveUp();

}

@Override

public void moveDown() {

topLeft.moveDown();

bottomRight.moveDown();

}

@Override

public void moveLeft() {

topLeft.moveLeft();

bottomRight.moveLeft();

}

@Override

public void moveRight() {

topLeft.moveRight();

bottomRight.moveRight();

}

@Override

public String toString() {

return "Rectangle[TopLeft=" + topLeft + ", BottomRight=" + bottomRight + "]";

}

}

public class MovableShapesSimulation {

public static void main(String[] args) {

MovablePoint p1 = new MovablePoint(0, 0, 2, 2);

System.out.println("initial point: " + p1);

p1.moveRight();

p1.moveUp();

System.out.println("after moving: " + p1);

MovableCircle circle = new MovableCircle(5, new MovablePoint(1, 1, 3, 3));

System.out.println("initial circle: " + circle);

circle.moveUp();

circle.moveLeft();

System.out.println("after moving: " + circle);

MovableRectangle rectangle = new MovableRectangle(

new MovablePoint(0, 0, 1, 1),

new MovablePoint(5, -5, 1, 1)

);

System.out.println("initial rectangle: " + rectangle);

rectangle.moveDown();

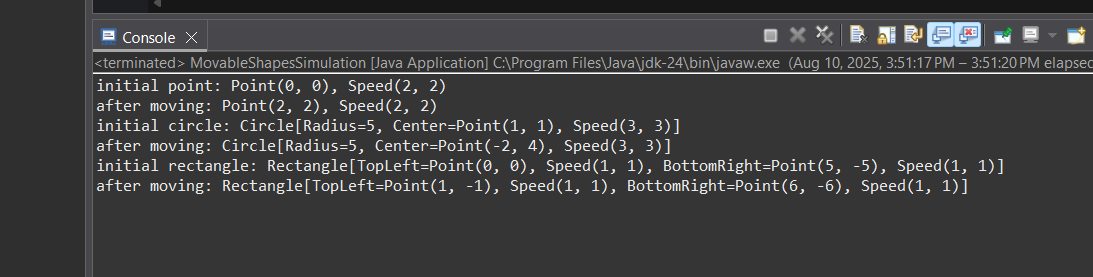
rectangle.moveRight();

System.out.println("after moving: " + rectangle);

}

}

**OUTPUT:**



1. **Answer**

interface Printer {

void print(String document);

}

class LaserPrinter implements Printer {

@Override

public void print(String document) {

System.out.println("Laser printer: " + document);

}

}

class InkjetPrinter implements Printer {

@Override

public void print(String document) {

System.out.println("Inkjet printer: " + document);

}

}

public class PrinterSwitchDemo {

public static void main(String[] args) {

Printer p;

p = new LaserPrinter();

p.print("annual report");

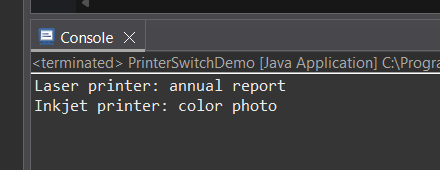
p = new InkjetPrinter();

p.print("color photo");

}

}

**OUTPUT:**

****

1. **Answer**

interface BaseVehicle {

void start();

}

interface AdvancedVehicle extends BaseVehicle {

void stop();

boolean refuel(int amount);

}

class Car implements AdvancedVehicle {

private int fuelLevel;

public Car(int fuelLevel) {

this.fuelLevel = fuelLevel;

}

@Override

public void start() {

if (fuelLevel > 0) {

System.out.println("car started fuel level: " + fuelLevel);

} else {

System.out.println("cannot start fuel is empty");

}

}

@Override

public void stop() {

System.out.println("car stopped");

}

@Override

public boolean refuel(int amount) {

if (amount > 0) {

fuelLevel += amount;

System.out.println("refueled " + amount + " unit current fuel: " + fuelLevel);

return true;

}

System.out.println("refuel amount should positive");

return false;

}

}

public class Main {

public static void main(String[] args) {

BaseVehicle baseRef = new Car(5);

baseRef.start();

System.out.println();

AdvancedVehicle advRef = new Car(0);

advRef.start();

advRef.refuel(10);

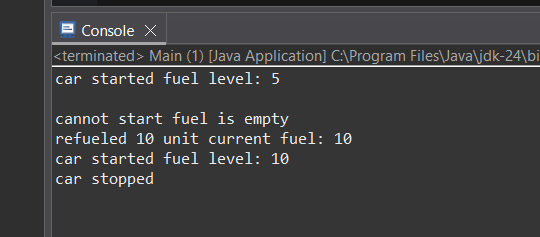
advRef.start();

advRef.stop();

}

}

**OUTPUT:**



1. **Answer**

import java.time.\*;

import java.util.ArrayList;

import java.util.List;

class TimeServer {

public static interface Client {

void updateTime(LocalDateTime now);

}

private final List<Client> clients = new ArrayList<>();

public void registerClient(Client client) {

clients.add(client);

}

public void notifyClients() {

LocalDateTime now = LocalDateTime.now();

for (Client client : clients) {

client.updateTime(now);

}

}

}

class ConsoleClock implements TimeServer.Client {

private final String name;

public ConsoleClock(String name) {

this.name = name;

}

@Override

public void updateTime(LocalDateTime now) {

System.out.println(name + " time: " + now);

}

}

class DigitalClock implements TimeServer.Client {

private final String location;

public DigitalClock(String location) {

this.location = location;

}

@Override

public void updateTime(LocalDateTime now) {

System.out.println("digital clock " + location + ": " + now);

}

}

public class Main2 {

public static void main(String[] args) throws InterruptedException {

TimeServer server = new TimeServer();

server.registerClient(new ConsoleClock("clock-1"));

server.registerClient(new DigitalClock(""));

for (int i = 0; i < 3; i++) {

server.notifyClients();

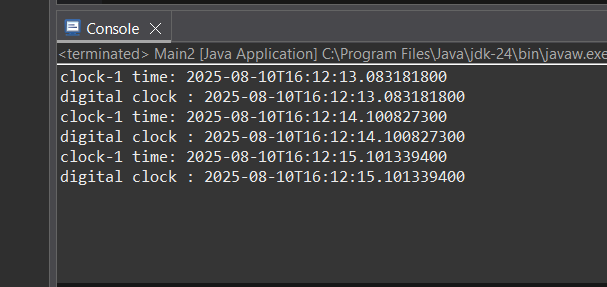
Thread.sleep(1000);

}

}

}

**OUTPUT:**



1. **Answer**

interface Polygon {

double getArea();

default double getPerimeter(int... sides) {

double perimeter = 0;

for (int side : sides) {

perimeter += side;

}

return perimeter;

}

static String shapeInfo() {

return "Polygon";

}

}

class Rectangle1 implements Polygon {

private double length;

private double width;

Rectangle1(double length, double width) {

this.length = length;

this.width = width;

}

*@Override*

public double getArea() {

return length \* width;

}

}

class Triangle implements Polygon {

private double base;

private double height;

Triangle(double base, double height) {

this.base = base;

this.height = height;

}

*@Override*

public double getArea() {

return 0.5 \* base \* height;

}

}

public class Main3 {

public static void main(String[] args) {

Rectangle1 rect = new Rectangle1(10, 5);

Triangle tri = new Triangle(6, 4);

System.***out***.println("rectangle area: " + rect.getArea());

System.***out***.println("rectangle perimeter: " + rect.getPerimeter(10, 5, 10, 5));

System.***out***.println("triangle area: " + tri.getArea());

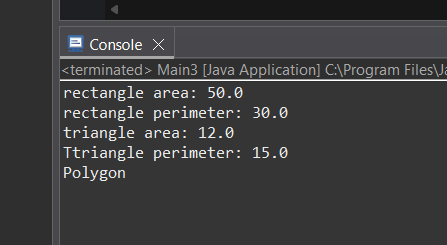
System.***out***.println("Ttriangle perimeter: " + tri.getPerimeter(6, 4, 5));

System.***out***.println(Polygon.*shapeInfo*());

}

}

**OUTPUT:**



**Lamda Expressions**

1. **Answer**

@FunctionalInterface

interface Sum {

int add(int a, int b);

}

public class LambdaSum {

public static void main(String[] args) {

Sum sum = (a, b) -> a + b;

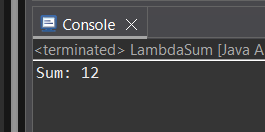
int result = sum.add(5, 7);

System.out.println("Sum: " + result);

}

}

**OUTPUT:**



1. **Answer**

@FunctionalInterface

interface SumCalculator {

int sum(int a, int b);

}

public class Mian4 {

public static void main(String[] args) {

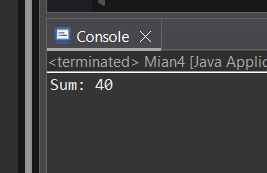
SumCalculator calculator = (a, b) -> a + b;

System.out.println("Sum: " + calculator.sum(15, 25));

}

}

**OUTPUT:**



1. **Answer**

import java.util.function.Predicate;

public class Empty {

public static void main(String[] args) {

Predicate<String> isEmpty = s -> s.isEmpty();

System.out.println(isEmpty.test(""));

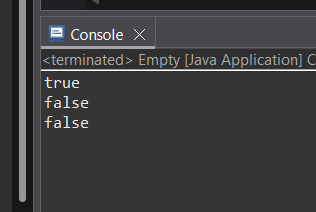
System.out.println(isEmpty.test("Hello"));

System.out.println(isEmpty.test(" "));

}

}

**OUTPUT:**



1. **Answer**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class EvenOrOdd {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(10, 15, 20, 25, 30, 35, 40);

List<Integer> evenNumbers = numbers.stream()

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

.collect(Collectors.toList());

List<Integer> oddNumbers = numbers.stream()

.filter(n -> n % 2 != 0)

.collect(Collectors.toList());

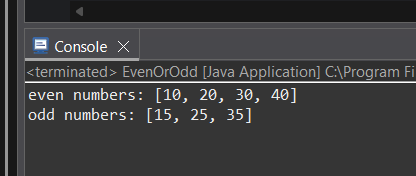
System.out.println("even numbers: " + evenNumbers);

System.out.println("odd numbers: " + oddNumbers);

}

}

**OUTPUT:**



1. **Answer**

import java.util.function.Function;

public class UpperCase {

public static void main(String[] args) {

Function<String, String> toUpperCase = s -> s.toUpperCase();

String original = "hello world";

String upper = toUpperCase.apply(original);

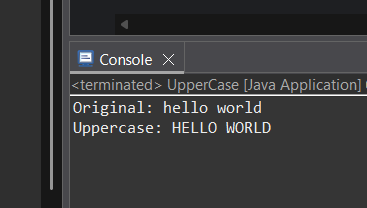
System.out.println("Original: " + original);

System.out.println("Uppercase: " + upper);

}

}

**OUTPUT:**



1. **Answer**

import java.util.\*;

public class SortStrings {

public static void main(String[] args) {

List<String> words = Arrays.asList("kiwi","pear", "banana", "strawberry", "apple");

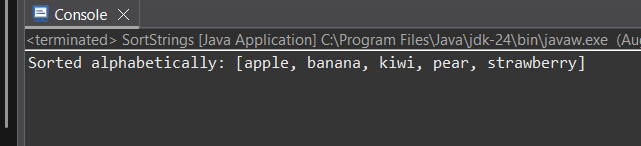
words.sort((s1, s2) -> s1.compareTo(s2));

System.out.println("Sorted alphabetically: " + words);

}

}

**OUTPUT:**



1. **Answer**

import java.util.Arrays;

import java.util.OptionalDouble;

public class Operations {

public static void main(String[] args) {

double[] numbers = {3.5, 7.2, 1.8, 9.0, 4.4};

double sum = Arrays.stream(numbers).sum();

OptionalDouble max = Arrays.stream(numbers).max();

OptionalDouble average = Arrays.stream(numbers).average();

System.out.println("sum: " + sum);

if (max.isPresent()) {

System.out.println("max: " + max.getAsDouble());

}

if (average.isPresent()) {

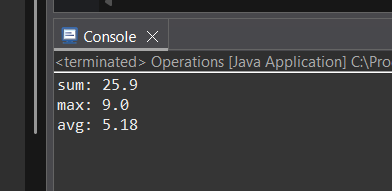
System.out.println("avg: " + average.getAsDouble());

}

}

}

**OUTPUT:**



1. **Answer**

@FunctionalInterface

interface TwoNumberOperation {

int operate(int a, int b);

}

public class MaxMin {

public static void main(String[] args) {

TwoNumberOperation max = (a, b) -> (a > b) ? a : b;

TwoNumberOperation min = (a, b) -> (a < b) ? a : b;

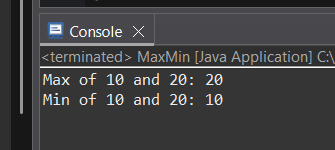
System.out.println("Max of 10 and 20: " + max.operate(10, 20));

System.out.println("Min of 10 and 20: " + min.operate(10, 20));

}

}

**OUTPUT:**



1. **Answer**

import java.util.function.IntFunction;

public class Factorial {

public static void main(String[] args) {

IntFunction<Long> factorial = n -> {

long result = 1;

for (int i = 2; i <= n; i++) {

result \*= i;

}

return result;

};

System.out.println("Factorial of 5: " + factorial.apply(5)); // 120

System.out.println("Factorial of 7: " + factorial.apply(7)); // 5040

}

}

**OUTPUT:**

