// 1. Print prime numbers up to a given number

public class PrimeNumbers {

public static void main(String[] args) {

int limit = 50;

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

boolean isPrime = true;

for (int j = 2; j <= Math.sqrt(i); j++) {

if (i % j == 0) {

isPrime = false;

break;

}

}

if (isPrime) System.out.println(i);

}

}

}

// 2. Add two double values using command-line arguments

public class AddDoubles {

public static void main(String[] args) {

double num1 = Double.parseDouble(args[0]);

double num2 = Double.parseDouble(args[1]);

System.out.println("Sum: " + (num1 + num2));

}

}

// 3. Illustrate static variables and methods

class StaticExample {

static int count = 0;

static void increment() {

count++;

}

public static void main(String[] args) {

increment();

increment();

System.out.println("Count: " + count);

}

}

// 4. Illustrate parameterized constructor and copy constructor

class Example {

int value;

Example(int value) {

this.value = value;

}

Example(Example obj) {

this.value = obj.value;

}

public static void main(String[] args) {

Example obj1 = new Example(10);

Example obj2 = new Example(obj1);

System.out.println("Value: " + obj2.value);

}

}

// 5. Illustrate "this" keyword

class ThisExample {

int value;

ThisExample(int value) {

this.value = value;

}

void display() {

System.out.println("Value: " + this.value);

}

public static void main(String[] args) {

ThisExample obj = new ThisExample(10);

obj.display();

}

}

// 6. Illustrate constructor overloading

class ConstructorOverloading {

ConstructorOverloading() {

System.out.println("Default Constructor");

}

ConstructorOverloading(int value) {

System.out.println("Parameterized Constructor: " + value);

}

public static void main(String[] args) {

new ConstructorOverloading();

new ConstructorOverloading(10);

}

}

// 7. Illustrate method overloading

class MethodOverloading {

void display(int value) {

System.out.println("Integer: " + value);

}

void display(String value) {

System.out.println("String: " + value);

}

public static void main(String[] args) {

MethodOverloading obj = new MethodOverloading();

obj.display(10);

obj.display("Hello");

}

}

// 8. Illustrate call by value and call by reference

class CallBy {

void callByValue(int value) {

value += 10;

System.out.println("Call By Value: " + value);

}

void callByReference(CallBy obj) {

obj.value += 10;

}

int value = 20;

public static void main(String[] args) {

CallBy obj = new CallBy();

obj.callByValue(obj.value);

obj.callByReference(obj);

System.out.println("Call By Reference: " + obj.value);

}

}

// 9. Find the average of array elements

public class AverageArray {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 4, 5};

int sum = 0;

for (int num : numbers) {

sum += num;

}

System.out.println("Average: " + (sum / (double) numbers.length));

}

}

// 10. Illustrate method overriding

class Parent {

void display() {

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

}

}

class Child extends Parent {

void display() {

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

}

public static void main(String[] args) {

Parent obj = new Child();

obj.display();

}

}

// 11. Demonstrate dynamic method dispatch

class Animal {

void sound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

void sound() {

System.out.println("Dog barks");

}

}

public class DynamicDispatch {

public static void main(String[] args) {

Animal obj = new Dog();

obj.sound();

}

}

// 12. Illustrate multilevel inheritance

class A {

void methodA() {

System.out.println("Method of Class A");

}

}

class B extends A {

void methodB() {

System.out.println("Method of Class B");

}

}

class C extends B {

void methodC() {

System.out.println("Method of Class C");

}

}

public class MultilevelInheritance {

public static void main(String[] args) {

C obj = new C();

obj.methodA();

obj.methodB();

obj.methodC();

}

}

// 13. Illustrate hierarchical inheritance

class X {

void methodX() {

System.out.println("Method of Class X");

}

}

class Y extends X {

void methodY() {

System.out.println("Method of Class Y");

}

}

class Z extends X {

void methodZ() {

System.out.println("Method of Class Z");

}

}

public class HierarchicalInheritance {

public static void main(String[] args) {

Y obj1 = new Y();

Z obj2 = new Z();

obj1.methodX();

obj1.methodY();

obj2.methodX();

obj2.methodZ();

}

}

// 14. Illustrate single inheritance

class ParentClass {

void parentMethod() {

System.out.println("Parent Class Method");

}

}

class ChildClass extends ParentClass {

void childMethod() {

System.out.println("Child Class Method");

}

}

public class SingleInheritance {

public static void main(String[] args) {

ChildClass obj = new ChildClass();

obj.parentMethod();

obj.childMethod();

}

}

// 15. Illustrate interface implementation

interface MyInterface {

void display();

}

class InterfaceExample implements MyInterface {

public void display() {

System.out.println("Interface Method Implemented");

}

public static void main(String[] args) {

MyInterface obj = new InterfaceExample();

obj.display();

}

}

// 16. Illustrate abstract class and method

abstract class AbstractClass {

abstract void abstractMethod();

void concreteMethod() {

System.out.println("Concrete Method");

}

}

class AbstractExample extends AbstractClass {

void abstractMethod() {

System.out.println("Abstract Method Implemented");

}

public static void main(String[] args) {

AbstractExample obj = new AbstractExample();

obj.abstractMethod();

obj.concreteMethod();

}

}

// 17. Illustrate encapsulation

class Encapsulation {

private int value;

public int getValue() {

return value;

}

public void setValue(int value) {

this.value = value;

}

public static void main(String[] args) {

Encapsulation obj = new Encapsulation();

obj.setValue(10);

System.out.println("Value: " + obj.getValue());

}

}

// 18. Illustrate polymorphism

class PolyExample {

void display() {

System.out.println("No arguments");

}

void display(int value) {

System.out.println("With arguments: " + value);

}

public static void main(String[] args) {

PolyExample obj = new PolyExample();

obj.display();

obj.display(5);

}

}

// 19. Reverse a string

public class ReverseString {

public static void main(String[] args) {

String str = "Hello";

String reversed = new StringBuilder(str).reverse().toString();

System.out.println("Reversed: " + reversed);

}

}

// 20. Check palindrome for a string

public class Palindrome {

public static void main(String[] args) {

String str = "radar";

String reversed = new StringBuilder(str).reverse().toString();

System.out.println(str.equals(reversed) ? "Palindrome" : "Not a Palindrome");

}

}

// 21. Illustrate threads using Runnable

class MyThread implements Runnable {

public void run() {

System.out.println("Thread Running");

}

}

public class ThreadExample {

public static void main(String[] args) {

Thread t = new Thread(new MyThread());

t.start();

}

}

// 22. Sort an array

import java.util.Arrays;

public class SortArray {

public static void main(String[] args) {

int[] arr = {5, 2, 8, 1, 3};

Arrays.sort(arr);

System.out.println("Sorted Array: " + Arrays.toString(arr));

}

}

// 23. Check Armstrong number

public class ArmstrongNumber {

public static void main(String[] args) {

int num = 153, sum = 0, temp = num;

while (temp > 0) {

int digit = temp % 10;

sum += Math.pow(digit, 3);

temp /= 10;

}

System.out.println(num == sum ? "Armstrong" : "Not Armstrong");

}

}

// 24. Fibonacci series

public class Fibonacci {

public static void main(String[] args) {

int n = 10, a = 0, b = 1;

System.out.print("Fibonacci: ");

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

System.out.print(a + " ");

int next = a + b;

a = b;

b = next;

}

}

}

// 25. Find factorial of a number

public class Factorial {

public static void main(String[] args) {

int num = 5, fact = 1;

for (int i = 1; i <= num; i++) fact \*= i;

System.out.println("Factorial: " + fact);

}

}

// 26. Find the sum of digits of a number

public class SumOfDigits {

public static void main(String[] args) {

int num = 123, sum = 0;

while (num > 0) {

sum += num % 10;

num /= 10;

}

System.out.println("Sum of Digits: " + sum);

}

}

// 27. Demonstrate try-catch block

public class TryCatchExample {

public static void main(String[] args) {

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

System.out.println("Exception: Division by zero");

}

}

}

// 28. Illustrate finally block

public class FinallyExample {

public static void main(String[] args) {

try {

int result = 10 / 2;

} finally {

System.out.println("Finally Block Executed");

}

}

}

// 29. Read from a file

import java.io.\*;

public class ReadFile {

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

BufferedReader br = new BufferedReader(new FileReader("example.txt"));

String line;

while ((line = br.readLine()) != null) {

System.out.println(line);

}

br.close();

}

}

// 30. Write to a file

import java.io.\*;

public class WriteFile {

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

BufferedWriter bw = new BufferedWriter(new FileWriter("example.txt"));

bw.write("Hello, World!");

bw.close();

}

}

// 31. Find the largest and smallest elements in an array

public class MinMaxArray {

public static void main(String[] args) {

int[] arr = {5, 1, 9, 3, 7};

int min = arr[0], max = arr[0];

for (int num : arr) {

if (num < min) min = num;

if (num > max) max = num;

}

System.out.println("Min: " + min + ", Max: " + max);

}

}

// 32. Create a thread using Thread class

class MyThreadClass extends Thread {

public void run() {

System.out.println("Thread Running");

}

}

public class ThreadUsingClass {

public static void main(String[] args) {

MyThreadClass t = new MyThreadClass();

t.start();

}

}

// 33. Matrix multiplication

public class MatrixMultiplication {

public static void main(String[] args) {

int[][] a = {{1, 2}, {3, 4}};

int[][] b = {{5, 6}, {7, 8}};

int[][] c = new int[2][2];

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

for (int k = 0; k < 2; k++) {

c[i][j] += a[i][k] \* b[k][j];

}

}

}

for (int[] row : c) {

for (int value : row) {

System.out.print(value + " ");

}

System.out.println();

}

}

}

// 34. Transpose of a matrix

public class TransposeMatrix {

public static void main(String[] args) {

int[][] matrix = {{1, 2, 3}, {4, 5, 6}};

int rows = matrix.length, cols = matrix[0].length;

int[][] transpose = new int[cols][rows];

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

for (int j = 0; j < cols; j++) {

transpose[j][i] = matrix[i][j];

}

}

for (int[] row : transpose) {

for (int value : row) {

System.out.print(value + " ");

}

System.out.println();

}

}

}

// 35. Implement bubble sort

public class BubbleSort {

public static void main(String[] args) {

int[] arr = {5, 3, 8, 4, 2};

for (int i = 0; i < arr.length - 1; i++) {

for (int j = 0; j < arr.length - 1 - i; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

for (int num : arr) System.out.print(num + " ");

}

}

// 36. Calculate GCD of two numbers

public class GCD {

public static void main(String[] args) {

int a = 54, b = 24;

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

System.out.println("GCD: " + a);

}

}

// 37. Calculate LCM of two numbers

public class LCM {

public static void main(String[] args) {

int a = 12, b = 15, gcd = 1;

for (int i = 1; i <= a && i <= b; i++) {

if (a % i == 0 && b % i == 0) gcd = i;

}

int lcm = (a \* b) / gcd;

System.out.println("LCM: " + lcm);

}

}

// 38. Swap two numbers without using a third variable

public class SwapNumbers {

public static void main(String[] args) {

int a = 5, b = 10;

a = a + b;

b = a - b;

a = a - b;

System.out.println("a: " + a + ", b: " + b);

}

}

// 39. Check if a number is prime

public class CheckPrime {

public static void main(String[] args) {

int num = 29;

boolean isPrime = true;

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) {

isPrime = false;

break;

}

}

System.out.println(num + " is " + (isPrime ? "Prime" : "Not Prime"));

}

}

// 40. Demonstrate file handling (read and write)

import java.io.\*;

public class FileHandling {

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

String filename = "sample.txt";

// Write to file

BufferedWriter writer = new BufferedWriter(new FileWriter(filename));

writer.write("Hello, File Handling!");

writer.close();

// Read from file

BufferedReader reader = new BufferedReader(new FileReader(filename));

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

reader.close();

}

}

// 41. Illustrate binary search

public class BinarySearch {

public static void main(String[] args) {

int[] arr = {2, 3, 4, 10, 40};

int key = 10, low = 0, high = arr.length - 1, mid;

boolean found = false;

while (low <= high) {

mid = (low + high) / 2;

if (arr[mid] == key) {

found = true;

System.out.println("Found at index: " + mid);

break;

} else if (arr[mid] < key) {

low = mid + 1;

} else {

high = mid - 1;

}

}

if (!found) System.out.println("Not Found");

}

}

// 42. Merge two arrays

public class MergeArrays {

public static void main(String[] args) {

int[] arr1 = {1, 3, 5};

int[] arr2 = {2, 4, 6};

int[] merged = new int[arr1.length + arr2.length];

System.arraycopy(arr1, 0, merged, 0, arr1.length);

System.arraycopy(arr2, 0, merged, arr1.length, arr2.length);

for (int num : merged) System.out.print(num + " ");

}

}

// 43. Illustrate Quick Sort

public class QuickSort {

static void quickSort(int[] arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

static int partition(int[] arr, int low, int high) {

int pivot = arr[high], i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

public static void main(String[] args) {

int[] arr = {10, 7, 8, 9, 1, 5};

quickSort(arr, 0, arr.length - 1);

for (int num : arr) System.out.print(num + " ");

}

}

// 44. Demonstrate stack operations using arrays

public class StackUsingArray {

int top = -1;

int[] stack = new int[5];

void push(int value) {

if (top == stack.length - 1) System.out.println("Stack Overflow");

else stack[++top] = value;

}

void pop() {

if (top == -1) System.out.println("Stack Underflow");

else System.out.println("Popped: " + stack[top--]);

}

void display() {

for (int i = 0; i <= top; i++) System.out.print(stack[i] + " ");

System.out.println();

}

public static void main(String[] args) {

StackUsingArray stack = new StackUsingArray();

stack.push(10); stack.push(20); stack.display(); stack.pop(); stack.display();

}

}

// 45. Demonstrate queue operations using arrays

public class QueueUsingArray {

int front = 0, rear = -1, size = 0;

int[] queue = new int[5];

void enqueue(int value) {

if (size == queue.length) System.out.println("Queue Overflow");

else {

rear = (rear + 1) % queue.length;

queue[rear] = value;

size++;

}

}

void dequeue() {

if (size == 0) System.out.println("Queue Underflow");

else {

System.out.println("Dequeued: " + queue[front]);

front = (front + 1) % queue.length;

size--;

}

}

void display() {

for (int i = 0; i < size; i++) System.out.print(queue[(front + i) % queue.length] + " ");

System.out.println();

}

public static void main(String[] args) {

QueueUsingArray queue = new QueueUsingArray();

queue.enqueue(10); queue.enqueue(20); queue.display(); queue.dequeue(); queue.display();

}

}

// 46. Reverse a string using recursion

public class ReverseString {

static String reverse(String str) {

if (str.isEmpty()) return str;

return reverse(str.substring(1)) + str.charAt(0);

}

public static void main(String[] args) {

String str = "Hello";

System.out.println("Reversed: " + reverse(str));

}

}

// 47. Illustrate singly linked list operations

class LinkedList {

Node head;

static class Node {

int data;

Node next;

Node(int data) { this.data = data; this.next = null; }

}

void add(int data) {

Node newNode = new Node(data);

if (head == null) head = newNode;

else {

Node temp = head;

while (temp.next != null) temp = temp.next;

temp.next = newNode;

}

}

void display() {

Node temp = head;

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.add(1); list.add(2); list.display();

}

}

// 48. Check if a string is a palindrome

public class Palindrome {

public static void main(String[] args) {

String str = "radar";

boolean isPalindrome = true;

for (int i = 0, j = str.length() - 1; i < j; i++, j--) {

if (str.charAt(i) != str.charAt(j)) {

isPalindrome = false;

break;

}

}

System.out.println(str + " is " + (isPalindrome ? "a palindrome" : "not a palindrome"));

}

}

// 49. Demonstrate binary tree operations

class BinaryTree {

Node root;

static class Node {

int data;

Node left, right;

Node(int data) { this.data = data; left = right = null; }

}

void inorder(Node node) {

if (node != null) {

inorder(node.left);

System.out.print(node.data + " ");

inorder(node.right);

}

}

public static void main(String[] args) {

BinaryTree tree = new BinaryTree();

tree.root = new Node(1);

tree.root.left = new Node(2);

tree.root.right = new Node(3);

tree.inorder(tree.root);

}

}

// 50. Implement a hash map

import java.util.HashMap;

public class HashMapExample {

public static void main(String[] args) {

HashMap<Integer, String> map = new HashMap<>();

map.put(1, "A"); map.put(2, "B");

System.out.println(map.get(1));

}

}

// 51. Demonstrate the use of enums

enum Day { MONDAY, TUESDAY, WEDNESDAY }

public class EnumExample {

public static void main(String[] args) {

Day day = Day.MONDAY;

System.out.println(day);

}

}

// 52. Find factorial using recursion

public class FactorialRecursion {

static int factorial(int n) {

if (n == 0) return 1;

return n \* factorial(n - 1);

}

public static void main(String[] args) {

int num = 5;

System.out.println("Factorial: " + factorial(num));

}

}EX