# Core Java – 41 Practical Exercises with Solutions

## 1. Hello World Program

Basic structure of a Java program that prints a greeting.

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

## 2. Simple Calculator

A four‑function calculator using user input.

import java.util.Scanner;

public class SimpleCalculator {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter first number: ");

double a = sc.nextDouble();

System.out.print("Enter second number: ");

double b = sc.nextDouble();

System.out.print("Choose operation (+ - \* /): ");

char op = sc.next().charAt(0);

double result;

switch (op) {

case '+': result = a + b; break;

case '-': result = a - b; break;

case '\*': result = a \* b; break;

case '/': result = b != 0 ? a / b : Double.NaN; break;

default: System.out.println("Invalid operator"); return;

}

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

}

}

## 3. Even or Odd Checker

Determines whether an integer is even or odd.

import java.util.Scanner;

public class EvenOdd {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter an integer: ");

int n = sc.nextInt();

System.out.println(n + " is " + (n % 2 == 0 ? "even" : "odd"));

}

}

## 4. Leap Year Checker

Checks whether a year is a leap year.

import java.util.Scanner;

public class LeapYear {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter year: ");

int y = sc.nextInt();

boolean leap = (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);

System.out.println(y + (leap ? " is " : " is not ") + "a leap year");

}

}

## 5. Multiplication Table

Prints multiplication table of a number up to 10.

import java.util.Scanner;

public class MultiplicationTable {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a number: ");

int n = sc.nextInt();

for (int i = 1; i <= 10; i++) {

System.out.printf("%d x %d = %d%n", n, i, n \* i);

}

}

}

## 6. Data Type Demonstration

Shows usage of Java primitive data types.

public class DataTypeDemo {

public static void main(String[] args) {

int i = 42;

float f = 3.14f;

double d = 2.718281828;

char c = 'J';

boolean b = true;

System.out.println("int: " + i);

System.out.println("float: " + f);

System.out.println("double: " + d);

System.out.println("char: " + c);

System.out.println("boolean: " + b);

}

}

## 7. Type Casting Example

Converts between double and int types.

public class TypeCasting {

public static void main(String[] args) {

double d = 9.99;

int i = (int) d; // narrowing

System.out.println("double to int: " + i);

int x = 7;

double y = x; // widening

System.out.println("int to double: " + y);

}

}

## 8. Operator Precedence

Demonstrates evaluation order in expressions.

public class OperatorPrecedence {

public static void main(String[] args) {

int result = 10 + 5 \* 2; // 10 + (5\*2) = 20

int result2 = (10 + 5) \* 2; // (10+5) \* 2 = 30

System.out.println("10 + 5 \* 2 = " + result);

System.out.println("(10 + 5) \* 2 = " + result2);

}

}

## 9. Grade Calculator

Assigns a letter grade based on numeric marks.

import java.util.Scanner;

public class GradeCalculator {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter marks (0‑100): ");

int m = sc.nextInt();

char grade;

if (m >= 90) grade = 'A';

else if (m >= 80) grade = 'B';

else if (m >= 70) grade = 'C';

else if (m >= 60) grade = 'D';

else grade = 'F';

System.out.println("Grade: " + grade);

}

}

## 10. Number Guessing Game

User guesses a randomly generated number between 1 and 100.

import java.util.Random;

import java.util.Scanner;

public class GuessingGame {

public static void main(String[] args) {

Random rand = new Random();

int target = rand.nextInt(100) + 1;

Scanner sc = new Scanner(System.in);

int guess;

do {

System.out.print("Guess a number (1‑100): ");

guess = sc.nextInt();

if (guess > target) System.out.println("Too high!");

else if (guess < target) System.out.println("Too low!");

} while (guess != target);

System.out.println("Correct! The number was " + target);

}

}

## 11. Factorial Calculator

Calculates factorial of a non‑negative integer.

import java.util.Scanner;

public class Factorial {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a non‑negative integer: ");

int n = sc.nextInt();

long fact = 1;

for (int i = 2; i <= n; i++) fact \*= i;

System.out.println(n + "! = " + fact);

}

}

## 12. Method Overloading

Adds numbers using overloaded methods.

public class OverloadAdd {

static int add(int a, int b) { return a + b; }

static double add(double a, double b) { return a + b; }

static int add(int a, int b, int c) { return a + b + c; }

public static void main(String[] args) {

System.out.println(add(2, 3));

System.out.println(add(2.5, 3.1));

System.out.println(add(1, 2, 3));

}

}

## 13. Recursive Fibonacci

Computes the nth Fibonacci number using recursion.

import java.util.Scanner;

public class RecursiveFibonacci {

static long fib(int n) {

if (n <= 1) return n;

return fib(n‑1) + fib(n‑2);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter n: ");

int n = sc.nextInt();

System.out.println("F(" + n + ") = " + fib(n));

}

}

## 14. Array Sum and Average

Calculates sum and average of an integer array.

import java.util.Scanner;

public class ArraySumAvg {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("How many elements? ");

int n = sc.nextInt();

int[] arr = new int[n];

int sum = 0;

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

System.out.print("Enter element " + (i + 1) + ": ");

arr[i] = sc.nextInt();

sum += arr[i];

}

double avg = (double) sum / n;

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

}

}

## 15. String Reversal

Reverses a user‑entered string.

import java.util.Scanner;

public class StringReverse {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a string: ");

String s = sc.nextLine();

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

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

}

}

## 16. Palindrome Checker

Checks whether a string is a palindrome.

import java.util.Scanner;

public class Palindrome {

static boolean isPalindrome(String s) {

s = s.replaceAll("[^A-Za-z0-9]", "").toLowerCase();

return new StringBuilder(s).reverse().toString().equals(s);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a string: ");

String s = sc.nextLine();

System.out.println(isPalindrome(s) ? "Palindrome" : "Not a palindrome");

}

}

## 17. Class and Object Creation

Defines a Car class and displays details.

class Car {

String make, model;

int year;

Car(String make, String model, int year) {

this.make = make; this.model = model; this.year = year;

}

void displayDetails() {

System.out.printf("Car: %s %s (%d)%n", make, model, year);

}

}

public class CarDemo {

public static void main(String[] args) {

Car car1 = new Car("Tesla", "Model S", 2024);

car1.displayDetails();

}

}

## 18. Inheritance Example

Demonstrates method overriding in subclasses.

class Animal {

void makeSound() { System.out.println("Some sound"); }

}

class Dog extends Animal {

@Override

void makeSound() { System.out.println("Bark"); }

}

public class InheritanceDemo {

public static void main(String[] args) {

Animal a = new Animal();

Animal d = new Dog();

a.makeSound();

d.makeSound();

}

}

## 19. Interface Implementation

Implements play() method in two musical instruments.

interface Playable {

void play();

}

class Guitar implements Playable {

public void play() { System.out.println("Strum strum"); }

}

class Piano implements Playable {

public void play() { System.out.println("Plink plonk"); }

}

public class InterfaceDemo {

public static void main(String[] args) {

Playable g = new Guitar();

Playable p = new Piano();

g.play();

p.play();

}

}

## 20. Try‑Catch Example

Handles division by zero.

import java.util.Scanner;

public class DivisionSafe {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter dividend: ");

int a = sc.nextInt();

System.out.print("Enter divisor: ");

int b = sc.nextInt();

try {

int res = a / b;

System.out.println("Result = " + res);

} catch (ArithmeticException ex) {

System.out.println("Cannot divide by zero!");

}

}

}

## 21. Custom Exception

Throws InvalidAgeException if age < 18.

class InvalidAgeException extends Exception {

InvalidAgeException(String msg) { super(msg); }

}

public class AgeCheck {

static void verify(int age) throws InvalidAgeException {

if (age < 18) throw new InvalidAgeException("Age must be 18 or above.");

}

public static void main(String[] args) {

try {

verify(16);

} catch (InvalidAgeException e) {

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

}

}

}

## 22. File Writing

Writes user input to output.txt.

import java.io.FileWriter;

import java.io.IOException;

import java.util.Scanner;

public class FileWriteDemo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter text: ");

String text = sc.nextLine();

try (FileWriter fw = new FileWriter("output.txt")) {

fw.write(text);

System.out.println("Written to output.txt");

} catch (IOException e) {

e.printStackTrace();

}

}

}

## 23. File Reading

Reads contents of output.txt.

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class FileReadDemo {

public static void main(String[] args) {

try (BufferedReader br = new BufferedReader(new FileReader("output.txt"))) {

String line;

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

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

## 24. ArrayList Example

Manages a list of student names.

import java.util.ArrayList;

import java.util.Scanner;

public class NameList {

public static void main(String[] args) {

ArrayList<String> names = new ArrayList<>();

Scanner sc = new Scanner(System.in);

String input;

while (true) {

System.out.print("Enter name (or 'quit'): ");

input = sc.nextLine();

if ("quit".equalsIgnoreCase(input)) break;

names.add(input);

}

System.out.println("Student names: " + names);

}

}

## 25. HashMap Example

Maps student IDs to names.

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class StudentMap {

public static void main(String[] args) {

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

Scanner sc = new Scanner(System.in);

while (true) {

System.out.print("ID (0 to stop): ");

int id = sc.nextInt(); sc.nextLine();

if (id == 0) break;

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

String name = sc.nextLine();

map.put(id, name);

}

System.out.print("Retrieve ID: ");

int id = sc.nextInt();

System.out.println("Name: " + map.get(id));

}

}

## 26. Thread Creation

Creates two concurrent threads.

class MsgThread extends Thread {

private final String msg;

MsgThread(String msg) { this.msg = msg; }

public void run() {

for (int i = 0; i < 5; i++) {

System.out.println(msg + " " + i);

}

}

}

public class ThreadDemo {

public static void main(String[] args) {

new MsgThread("Hello").start();

new MsgThread("World").start();

}

}

## 27. Lambda Expressions

Sorts a list of strings using lambda.

import java.util.Arrays;

import java.util.List;

import java.util.Collections;

public class LambdaSort {

public static void main(String[] args) {

List<String> list = Arrays.asList("banana", "apple", "cherry");

Collections.sort(list, (a, b) -> a.compareToIgnoreCase(b));

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

}

}

## 28. Stream API

Filters even numbers from a list using streams.

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class StreamEven {

public static void main(String[] args) {

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

List<Integer> evens = nums.stream()

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

.collect(Collectors.toList());

System.out.println(evens);

}

}

## 29. Records (Java 16+)

Defines immutable data with record keyword.

public record Person(String name, int age) {}

import java.util.List;

public class RecordDemo {

public static void main(String[] args) {

List<Person> people = List.of(new Person("Alice", 30),

new Person("Bob", 20));

people.stream()

.filter(p -> p.age() >= 25)

.forEach(System.out::println);

}

}

## 30. Pattern Matching for switch (Java 21)

Uses pattern matching in enhanced switch expressions.

public class TypeSwitch {

static void test(Object obj) {

switch (obj) {

case Integer i -> System.out.println("Integer: " + i);

case String s -> System.out.println("String: " + s.toUpperCase());

case Double d -> System.out.println("Double: " + d);

default -> System.out.println("Unknown type");

}

}

public static void main(String[] args) {

test(42);

test("hello");

test(3.14);

}

}

## 31. Basic JDBC Connection

Connects to a SQLite DB and retrieves student data.

import java.sql.\*;

public class JDBCSelect {

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

Class.forName("org.sqlite.JDBC");

try (Connection con = DriverManager.getConnection("jdbc:sqlite:school.db");

Statement st = con.createStatement();

ResultSet rs = st.executeQuery("SELECT id, name FROM students")) {

while (rs.next()) {

System.out.printf("%d: %s%n", rs.getInt("id"), rs.getString("name"));

}

}

}

}

## 32. Insert and Update Operations in JDBC

Performs insert and update using PreparedStatement.

import java.sql.\*;

public class StudentDAO {

private final Connection con;

public StudentDAO(Connection con) { this.con = con; }

public void insert(int id, String name) throws SQLException {

try (PreparedStatement ps = con.prepareStatement(

"INSERT INTO students(id, name) VALUES (?, ?)")) {

ps.setInt(1, id); ps.setString(2, name);

ps.executeUpdate();

}

}

public void update(int id, String name) throws SQLException {

try (PreparedStatement ps = con.prepareStatement(

"UPDATE students SET name = ? WHERE id = ?")) {

ps.setString(1, name); ps.setInt(2, id);

ps.executeUpdate();

}

}

}

## 33. Transaction Handling in JDBC

Simulates money transfer between two accounts.

import java.sql.\*;

public class TransferService {

private final Connection con;

public TransferService(Connection con) { this.con = con; }

public void transfer(int from, int to, double amount) throws SQLException {

con.setAutoCommit(false);

try (PreparedStatement debit = con.prepareStatement(

"UPDATE accounts SET balance = balance - ? WHERE id = ?");

PreparedStatement credit = con.prepareStatement(

"UPDATE accounts SET balance = balance + ? WHERE id = ?")) {

debit.setDouble(1, amount);

debit.setInt(2, from);

credit.setDouble(1, amount);

credit.setInt(2, to);

debit.executeUpdate();

credit.executeUpdate();

con.commit();

} catch (SQLException e) {

con.rollback();

throw e;

} finally {

con.setAutoCommit(true);

}

}

}

## 34. Create and Use Java Modules

Sets up two modules and shares a utility class.

// com.utils/module-info.java

module com.utils {

exports com.utils;

}

// com.utils/com/utils/StringUtil.java

package com.utils;

public class StringUtil {

public static String shout(String s) { return s.toUpperCase() + "!"; }

}

// com.greetings/module-info.java

module com.greetings {

requires com.utils;

}

// com.greetings/com/greetings/Hello.java

package com.greetings;

import com.utils.StringUtil;

public class Hello {

public static void main(String[] args) {

System.out.println(StringUtil.shout("hello modules"));

}

}

## 35. TCP Client‑Server Chat

A rudimentary two‑way chat over TCP.

// Server

import java.io.\*;

import java.net.\*;

public class ChatServer {

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

try (ServerSocket ss = new ServerSocket(5000);

Socket client = ss.accept();

BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));

PrintWriter out = new PrintWriter(client.getOutputStream(), true);

BufferedReader console = new BufferedReader(new InputStreamReader(System.in))) {

String msg;

while ((msg = in.readLine()) != null) {

System.out.println("Client: " + msg);

out.println(console.readLine());

}

}

}

}

// Client

import java.io.\*;

import java.net.\*;

public class ChatClient {

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

try (Socket s = new Socket("localhost", 5000);

BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));

PrintWriter out = new PrintWriter(s.getOutputStream(), true);

BufferedReader console = new BufferedReader(new InputStreamReader(System.in))) {

String msg;

while ((msg = console.readLine()) != null) {

out.println(msg);

System.out.println("Server: " + in.readLine());

}

}

}

}

## 36. HTTP Client API (Java 11+)

Fetches JSON from a public API.

import java.net.http.\*;

import java.net.URI;

public class HttpDemo {

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

HttpClient client = HttpClient.newHttpClient();

HttpRequest req = HttpRequest.newBuilder()

.uri(URI.create("https://api.github.com/"))

.build();

HttpResponse<String> res = client.send(req, HttpResponse.BodyHandlers.ofString());

System.out.println("Status: " + res.statusCode());

System.out.println(res.body());

}

}

## 37. Using javap to Inspect Bytecode

Shows how to inspect class bytecode.

// Compile a class then run: javap -c ClassName

public class BytecodeDemo {

public static int square(int x) { return x \* x; }

}

## 38. Decompile a Class File

Steps to decompile using CFR or JD‑GUI.

// 1. Compile HelloWorld.java -> HelloWorld.class

// 2. Open the .class file in JD‑GUI or run: java -jar cfr.jar HelloWorld.class

// 3. View the reconstructed source code.

## 39. Reflection in Java

Loads a class and invokes its methods dynamically.

public class ReflectionDemo {

public void greet() { System.out.println("Hello via reflection"); }

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

Class<?> cls = Class.forName("ReflectionDemo");

Object obj = cls.getDeclaredConstructor().newInstance();

cls.getMethod("greet").invoke(obj);

}

}

## 40. Virtual Threads (Java 21)

Launches 100,000 virtual threads.

public class VirtualThreads {

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

for (int i = 0; i < 100\_000; i++) {

Thread.startVirtualThread(() -> System.out.println("Hi from " + Thread.currentThread()));

}

Thread.sleep(1000); // wait for threads

}

}

## 41. Executor Service and Callable

Runs multiple Callable tasks and collects results.

import java.util.concurrent.\*;

import java.util.List;

public class CallableDemo {

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

ExecutorService ex = Executors.newFixedThreadPool(4);

List<Callable<Integer>> tasks = List.of(

() -> 1 + 1,

() -> 2 + 2,

() -> 3 + 3

);

List<Future<Integer>> results = ex.invokeAll(tasks);

for (Future<Integer> f : results) {

System.out.println("Result: " + f.get());

}

ex.shutdown();

}

}