1. Write a program to:
   * Read an int value from user input.
   * Assign it to a double (implicit widening) and print both.
   * Read a double, explicitly cast it to int, then to short, and print results—demonstrate truncation or overflow.

ANS:

import java.util.Scanner;

public class TypeCastingDemo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int intValue = sc.nextInt();

double widenedValue = intValue;

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

System.out.println("Widened to double: " + widenedValue);

System.out.print("Enter a double value: ");

double doubleValue = sc.nextDouble();

int intFromDouble = (int) doubleValue;

short shortFromDouble = (short) intFromDouble;

System.out.println("Original double value: " + doubleValue);

System.out.println("After casting to int: " + intFromDouble);

System.out.println("After casting to short: " + shortFromDouble);

sc.close();

}

}

2.Convert an int to String using String.valueOf(...), then back with Integer.parseInt(...). Handle NumberFormatException.

ANS:

public class IntStringConversion {

public static void main(String[] args) {

int number = 123;

String strNumber = String.valueOf(number);

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

try {

int parsedNumber = Integer.parseInt(strNumber);

System.out.println("Parsed int value: " + parsedNumber);

String invalidStr = "123abc";

int invalidParsed = Integer.parseInt(invalidStr);

System.out.println("This won't print: " + invalidParsed);

} catch (NumberFormatException e) {

System.out.println("Error: Invalid number format - " + e.getMessage());

}

}

}

3.Compound Assignment Behaviour

1. Initialize int x = 5;.
2. Write two operations:

x = x + 4.5;

x += 4.5;

1. Print results and explain behavior in comments (implicit narrowing, compile error vs. successful assignment).

ANS:  
public class CompoundAssignmentDemo {

public static void main(String[] args) {

int x = 5;

x = (int) (x + 4.5); // Explicit narrowing from double → int

System.out.println("After explicit cast (x = x + 4.5): " + x); // Output: 9 (truncated)

x = 5;

x += 4.5;

System.out.println("After compound assignment (x += 4.5): " + x); // Output: 9

}

}

4. Object Casting with Inheritance

1. Define an Animal class with a method makeSound().
2. Define subclass Dog:
   * Override makeSound() (e.g. "Woof!").
   * Add method fetch().
3. In main:

Dog d = new Dog();

Animal a = d; // upcasting

a.makeSound();

ANS:

// Parent class

class Animal {

void makeSound() {

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

}

}

// Subclass

class Dog extends Animal {

@Override

void makeSound() {

System.out.println("Woof!");

}

void fetch() {

System.out.println("Dog is fetching the ball...");

}

}

public class CastingDemo {

public static void main(String[] args) {

Dog d = new Dog();

Animal a = d;

a.makeSound();

Dog d2 = (Dog) a;

d2.fetch();

}

}

5. Mini‑Project – Temperature Converter

1. Prompt user for a temperature in Celsius (double).
2. Convert it to Fahrenheit:

double fahrenheit = celsius \* 9/5 + 32;

1. Then cast that fahrenheit to int for display.
2. Print both the precise (double) and truncated (int) values, and comment on precision loss.

ANS:

import java.util.Scanner;

public class TemperatureConverter {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter temperature in Celsius: ");

double celsius = sc.nextDouble();

double fahrenheit = celsius \* 9 / 5 + 32;

int fahrenheitInt = (int) fahrenheit;

System.out.println("Fahrenheit (precise) : " + fahrenheit);

System.out.println("Fahrenheit (truncated): " + Fahrenheit);

}

}

6.Enum

1: Days of the Week

Define an enum DaysOfWeek with seven constants. Then in main(), prompt the user to input a day name and:

Print its position via ordinal().

Confirm if it's a weekend day using a switch or if-statement.

ANS:  
import java.util.Scanner;

public class EnumExample {

enum DaysOfWeek {

MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a day name (e.g., MONDAY): ");

String input = sc.next().toUpperCase();

try {

DaysOfWeek day = DaysOfWeek.valueOf(input);

System.out.println("Position (ordinal): " + day.ordinal());

switch (day) {

case SATURDAY:

case SUNDAY:

System.out.println(day + " is a weekend day!");

break;

default:

System.out.println(day + " is a weekday.");

}

} catch (IllegalArgumentException e) {

System.out.println("Invalid day entered. Please enter a valid day name.");

}

}

}

7.Compass Directions

Create an enum Direction with the values NORTH, SOUTH, EAST, WEST. Write code to:

* Read a Direction from a string using valueOf().

Use switch or if to print movement (e.g. “Move north”).  
Test invalid inputs with proper error handling.

ANS:  
import java.util.Scanner;

public class CompassExample {

enum Direction {

NORTH, SOUTH, EAST, WEST

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a direction (NORTH, SOUTH, EAST, WEST): ");

String input = sc.next().toUpperCase();

try {

Direction dir = Direction.valueOf(input);

switch (dir) {

case NORTH:

System.out.println("Move north");

break;

case SOUTH:

System.out.println("Move south");

break;

case EAST:

System.out.println("Move east");

break;

case WEST:

System.out.println("Move west");

break;

}

} catch (IllegalArgumentException e) {

System.out.println("Invalid direction! Please enter NORTH, SOUTH, EAST, or WEST.");

}

}

}

8.Shape Area Calculator

Define enum Shape (CIRCLE, SQUARE, RECTANGLE, TRIANGLE) where each constant:

* Overrides a method double area(double... params) to compute its area.
* E.g., CIRCLE expects radius, TRIANGLE expects base and height.  
  Loop over all constants with sample inputs and print results.

ANS:

public class ShapeAreaCalculator {

// 1. Define enum with abstract method

enum Shape {

CIRCLE {

@Override

double area(double... params) {

if (params.length < 1) throw new IllegalArgumentException("Circle needs radius");

double radius = params[0];

return Math.PI \* radius \* radius;

}

},

SQUARE {

@Override

double area(double... params) {

if (params.length < 1) throw new IllegalArgumentException("Square needs side length");

double side = params[0];

return side \* side;

}

},

RECTANGLE {

@Override

double area(double... params) {

if (params.length < 2) throw new IllegalArgumentException("Rectangle needs length and width");

double length = params[0];

double width = params[1];

return length \* width;

}

},

TRIANGLE {

@Override

double area(double... params) {

if (params.length < 2) throw new IllegalArgumentException("Triangle needs base and height");

double base = params[0];

double height = params[1];

return 0.5 \* base \* height;

}

};

abstract double area(double... params);

}

public static void main(String[] args) {

for (Shape shape : Shape.values()) {

double result;

switch (shape) {

case CIRCLE:

result = shape.area(5); // radius

break;

case SQUARE:

result = shape.area(4); // side

break;

case RECTANGLE:

result = shape.area(4, 6); // length, width

break;

case TRIANGLE:

result = shape.area(3, 8); // base, height

break;

default:

throw new IllegalStateException("Unexpected shape: " + shape);

}

System.out.printf("%s area = %.2f%n", shape, result);

}

}

}

9.Card Suit & Rank

Redesign a Card class using two enums: Suit (CLUBS, DIAMONDS, HEARTS, SPADES) and Rank (ACE…KING).  
Then implement a Deck class to:

Create all 52 cards.

Shuffle and print the order.

ANS:  
import java.util.\*;

// Enum for suits

enum Suit {

CLUBS, DIAMONDS, HEARTS, SPADES

}

enum Rank {

ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING

}

// Card class using enums

class Card {

private final Suit suit;

private final Rank rank;

public Card(Suit suit, Rank rank) {

this.suit = suit;

this.rank = rank;

}

public Suit getSuit() {

return suit;

}

public Rank getRank() {

return rank;

}

@Override

public String toString() {

return rank + " of " + suit;

}

}

// Deck class

class Deck {

private final List<Card> cards = new ArrayList<>();

public Deck() {

// Create all 52 cards

for (Suit suit : Suit.values()) {

for (Rank rank : Rank.values()) {

cards.add(new Card(suit, rank));

}

}

}

public void shuffle() {

Collections.shuffle(cards);

}

public void printDeck() {

for (Card card : cards) {

System.out.println(card);

}

}

}

// Main class

public class CardGame {

public static void main(String[] args) {

Deck deck = new Deck();

System.out.println("Before shuffling:");

deck.printDeck();

deck.shuffle();

System.out.println("\nAfter shuffling:");

deck.printDeck();

}

}

10. Priority Levels with Extra Data

Implement enum PriorityLevel with constants (LOW, MEDIUM, HIGH, CRITICAL), each having:

A numeric severity code.

A boolean isUrgent() if severity ≥ some threshold.  
Print descriptions and check urgency.

ANS:  
// Enum with extra fields and methods

enum PriorityLevel {

LOW(1),

MEDIUM(2),

HIGH(3),

CRITICAL(4);

private final int severityCode;

PriorityLevel(int severityCode) {

this.severityCode = severityCode;

}

public int getSeverityCode() {

return severityCode;

}

public boolean isUrgent() {

return severityCode >= 3; // HIGH and CRITICAL are urgent

}

@Override

public String toString() {

return name() + " (Severity " + severityCode + ")";

}

}

public class PriorityTest {

public static void main(String[] args) {

for (PriorityLevel level : PriorityLevel.values()) {

System.out.println(level +

" | Urgent? " + level.isUrgent());

}

}

}

11. Traffic Light State Machine

Implement enum TrafficLight implementing interface State, with constants RED, GREEN, YELLOW.  
Each must override State next() to transition in the cycle.  
Simulate and print six transitions starting from RED.

ANS:

// Interface defining state transition

interface State {

State next();

}

enum TrafficLight implements State {

RED {

@Override

public State next() {

return GREEN;

}

},

GREEN {

@Override

public State next() {

return YELLOW;

}

},

YELLOW {

@Override

public State next() {

return RED;

}

};

}

public class TrafficLightSimulation {

public static void main(String[] args) {

State current = TrafficLight.RED; // Starting point

for (int i = 0; i < 6; i++) {

System.out.println("Current Light: " + current);

current = current.next(); // Move to the next state

}

}}

12. Difficulty Level & Game Setup

Define enum Difficulty with EASY, MEDIUM, HARD.  
Write a Game class that takes a Difficulty and prints logic like:

EASY → 3000 bullets, MEDIUM → 2000, HARD → 1000.  
Use a switch(diff) inside constructor or method.

ANS:

enum Difficulty {

EASY, MEDIUM, HARD

}

class Game {

private int bullets;

public Game(Difficulty diff) {

switch (diff) {

case EASY:

bullets = 3000;

break;

case MEDIUM:

bullets = 2000;

break;

case HARD:

bullets = 1000;

break;

default:

throw new IllegalArgumentException("Unknown difficulty: " + diff);

}

System.out.println("Game started with difficulty: " + diff + " → Bullets: " + bullets);

}

}

public class GameSetup {

public static void main(String[] args) {

new Game(Difficulty.EASY);

new Game(Difficulty.MEDIUM);

new Game(Difficulty.HARD);

}

}

13. Calculator Operations Enum

Create enum Operation (PLUS, MINUS, TIMES, DIVIDE) with an eval(double a, double b) method.  
Implement two versions:

One using a switch(this) inside eval.

Another using constant-specific method overrides for eval.  
Compare both designs.

ANS:

INSIDE EVAL:  
enum OperationSwitch {

PLUS, MINUS, TIMES, DIVIDE;

public double eval(double a, double b) {

switch (this) {

case PLUS:

return a + b;

case MINUS:

return a - b;

case TIMES:

return a \* b;

case DIVIDE:

if (b == 0) throw new ArithmeticException("Division by zero");

return a / b;

default:

throw new AssertionError("Unknown operation: " + this);

}

}

}

public class CalculatorSwitch {

public static void main(String[] args) {

System.out.println("Switch version:");

for (OperationSwitch op : OperationSwitch.values()) {

System.out.printf("10 %s 5 = %.2f%n", op, op.eval(10, 5));

}

}

}

CONSTANT METHOD:

enum OperationOverride {

PLUS {

public double eval(double a, double b) { return a + b; }

},

MINUS {

public double eval(double a, double b) { return a - b; }

},

TIMES {

public double eval(double a, double b) { return a \* b; }

},

DIVIDE {

public double eval(double a, double b) {

if (b == 0) throw new ArithmeticException("Division by zero");

return a / b;

}

};

public abstract double eval(double a, double b);

}

public class CalculatorOverride {

public static void main(String[] args) {

System.out.println("\nOverride version:");

for (OperationOverride op : OperationOverride.values()) {

System.out.printf("10 %s 5 = %.2f%n", op, op.eval(10, 5));

}

}

}

14. Knowledge Level from Score Range

Define enum KnowledgeLevel with constants BEGINNER, ADVANCED, PROFESSIONAL, MASTER.  
Use a static method fromScore(int score) to return the appropriate enum:

0–3 → BEGINNER, 4–6 → ADVANCED, 7–9 → PROFESSIONAL, 10 → MASTER.

ANS:  
enum KnowledgeLevel {

BEGINNER, ADVANCED, PROFESSIONAL, MASTER;

public static KnowledgeLevel fromScore(int score) {

if (score >= 0 && score <= 3) {

return BEGINNER;

} else if (score >= 4 && score <= 6) {

return ADVANCED;

} else if (score >= 7 && score <= 9) {

return PROFESSIONAL;

} else if (score == 10) {

return MASTER;

} else {

throw new IllegalArgumentException("Invalid score: " + score);

}

}

}

public class KnowledgeTest {

public static void main(String[] args) {

int[] scores = {0, 3, 4, 6, 7, 9, 10};

for (int score : scores) {

KnowledgeLevel level = KnowledgeLevel.fromScore(score);

System.out.printf("Score: %d → Level: %s%n", score, level);

}

}

}

15. Exception handling

1: Division & Array Access

Write a Java class ExceptionDemo with a main method that:

1. Attempts to divide an integer by zero and access an array out of bounds.
2. Wrap each risky operation in its own try‑catch:
   * Catch only the specific exception types: ArithmeticException and ArrayIndexOutOfBoundsException.
   * In each catch, print a user-friendly message.
3. Add a finally block after each try‑catch that prints "Operation completed.".

Example structure:

try {

} catch (ArithmeticException e) {

System.out.println("Division by zero is not allowed!");

} finally {

System.out.println("Operation completed.");

}

ANS:

public class ExceptionDemo {

public static void main(String[] args) {

try {

int a = 10;

int b = 0;

int result = a / b;

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

} catch (ArithmeticException e) {

System.out.println("Division by zero is not allowed!");

} finally {

System.out.println("Operation completed.");

}

System.out.println("-----------------------");

try {

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

System.out.println(numbers[5]);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Invalid array index accessed!");

} finally {

System.out.println("Operation completed.");

}

}

}

16. Throw and Handle Custom Exception

Create a class OddChecker:

1. Implement a static method:

public static void checkOdd(int n) throws OddNumberException { /\* ... \*/ }

1. If n is odd, throw a custom checked exception OddNumberException with message "Odd number: " + n.
2. In main:
   * Call checkOdd with different values (including odd and even).
   * Handle exceptions with try‑catch, printing e.getMessage() when caught.

Define the exception like:

public class OddNumberException extends Exception {

public OddNumberException(String message) { super(message); }

}

ANS:

public class OddNumberException extends Exception {

public OddNumberException(String message) {

super(message);

}

}

public class OddChecker {

public static void checkOdd(int n) throws OddNumberException {

if (n % 2 != 0) {

throw new OddNumberException("Odd number: " + n);

} else {

System.out.println(n + " is even.");

}

}

public static void main(String[] args) {

int[] numbers = {2, 5, 8, 11, 14};

for (int num : numbers) {

try {

checkOdd(num);

} catch (OddNumberException e) {

System.out.println("Caught Exception → " + e.getMessage());

}

}

}

}

17. File Handling with Multiple Catches

Create a class FileReadDemo:

1. In main, call a method readFile(String filename) that declares throws FileNotFoundException, IOException.
2. In readFile, use FileReader (or BufferedReader) to open and read the first line of the file.
3. Handle exceptions in main using separate catch blocks:
   * catch (FileNotFoundException e) → print "File not found: " + filename
   * catch (IOException e) → print "Error reading file: " + e.getMessage()"
4. Include a finally block that prints "Cleanup done." regardless of outcome.

ANS:

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class FileReadDemo {

// Method to read first line from a file

public static void readFile(String filename) throws FileNotFoundException, IOException {

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

String firstLine = br.readLine();

System.out.println("First line of file: " + firstLine);

br.close();

}

public static void main(String[] args) {

String filename = "sample.txt"; // Change to your file path

try {

readFile(filename);

}

catch (FileNotFoundException e) {

System.out.println("File not found: " + filename);

}

catch (IOException e) {

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

}

finally {

System.out.println("Cleanup done.");

}

}

}

19. Multi‑Exception in One Try Block

Write a class MultiExceptionDemo:

* In a single try block, perform:
  + Opening a file
  + Parsing its first line as integer
  + Dividing 100 by that integer
* Use multiple catch blocks in this order:
  + FileNotFoundException
  + IOException
  + NumberFormatException
  + ArithmeticException
* In each catch, print a tailored message:
  + File not found
  + Problem reading file
  + Invalid number format
  + Division by zero
* Finally, print "Execution completed".

ANS:  
import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class MultiExceptionDemo {

public static void main(String[] args) {

String filename = "numbers.txt"; // Change to your file path

try {

// Open file

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

// Read first line

String firstLine = br.readLine();

br.close();

int number = Integer.parseInt(firstLine);

int result = 100 / number;

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

}

catch (FileNotFoundException e) {

System.out.println("File not found");

}

catch (IOException e) {

System.out.println("Problem reading file");

}

catch (NumberFormatException e) {

System.out.println("Invalid number format");

}

catch (ArithmeticException e) {

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

}

finally {

System.out.println("Execution completed");

}

}

}