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

Compound Assignment Behaviour

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

x = x + 4.5; // Does this compile? Why or why not?

x += 4.5; // What happens here?

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

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();

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.

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.

2: 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.

3: 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.

4.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.

5: 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.

6: 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.

7: 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.

8: 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.

10: 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.  
  Then print the level and test boundary conditions.

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 {

// division or array access

} catch (ArithmeticException e) {

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

} finally {

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

}

2: 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); }

}

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.

4: 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".