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.

Code; **package** Type\_casting;

**import** java.util.\*;

**public** **class** implicit\_casting {

**public** **static** **void** main(String[] args) {

Scanner scan=**new** Scanner(System.***in***);

System.***out***.println("Enter a integer number:");

**int** n=scan.nextInt();

**double** d=n;

System.***out***.println("Entered integer value is :"+n);

System.***out***.println("Entered double value after implicit widening is :"+d);

System.***out***.println("Enter a double number:");

**double** b=scan.nextDouble();

**int** q=(**int**) b;

**short** s = (**short**) q;

System.***out***.println("Entered double value: " + b);

System.***out***.println("Explicitly cast to int (truncated): " + q);

System.***out***.println("Then cast to short (possible overflow): " + s);

scan.close();

}

}

Output; Enter a integer number:

6

Entered integer value is :6

Entered double value after implicit widening is :6.0

Enter a double number:

5.0

Entered double value: 5.0

Explicitly cast to int (truncated): 5

Then cast to short (possible overflow): 5

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

Code; **package** Type\_casting;

**import** java.util.\*;

**public** **class** String\_int {

**public** **static** **void** main(String[] args) {

Scanner scan=**new** Scanner(System.***in***);

**try** {

System.***out***.println("Enter a integer number:");

**int** n=scan.nextInt();

String s=String.*valueOf*(n);

System.***out***.println("Entered integer value is:"+n);

System.***out***.println("Entered String AFTER CONVERTING INTEGER value is:"+s);

**int** q=Integer.*parseInt*(s);

System.***out***.println("Entered integer value after conerting string into int is:"+q);

}

**catch**(NumberFormatException e) {

e.printStackTrace();

}**finally** {

scan.close();

}

}

}

Output; Enter a integer number:

5

Entered integer value is:5

Entered String AFTER CONVERTING INTEGER value is:5

Entered integer value after conerting string into int is:5

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?

Code; **package** Type\_casting;

**import** java.util.\*;

**public** **class** Compound\_assignment\_behaviour {

**public** **static** **void** main(String[] args) {

**try**{

**int** x=5;

// This line would cause a compile-time error:

//x=x+4.5; // Error: cannot assign double to int without casting

// Compound assignment works due to implicit narrowing:

x+=4.5; // Equivalent to: x = (int)(x + 4.5)

System.***out***.println("Entered x value is "+x);

}**catch**(NumberFormatException e) {

e.printStackTrace();

}

}

}

Output;Entered x value is 9

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();

code; **package** Type\_casting;

**class** Animal1 {

**public** **void** makeSound() {

System.***out***.println("Making sound.");

}

}

**class** Dog **extends** Animal1{

**public** **void** makeSound() {

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

}

**public** **void** fetch() {

System.***out***.println("Fetching the ball!");

}

}

**public** **class** Animal {

**public** **static** **void** main(String[] args) {

Dog d = **new** Dog(); // Create Dog object

Animal1 a = d; // Upcasting: Dog → Animal

a.makeSound(); // Calls Dog's overridden method

}

}

Output; Woof!

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.

Code; **package** Type\_casting;

**import** java.util.\*;

**public** **class** Temperature\_converter {

**public** **static** **void** main(String[] args) {

Scanner scan=**new** Scanner(System.***in***);

System.***out***.println("Enter temperature in celsius:");

**double** celsius=scan.nextDouble();

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

**int** n=(**int**) fahrenheit;

System.***out***.println("Temperature in fahrenheit is:"+fahrenheit);

System.***out***.println("Temperature converting into integer value:"+n);

// Comment on precision loss

System.***out***.println("Note: Casting to int removes the decimal part, which may lead to loss of precision.");

scan.close();

}

}

Output; Enter temperature in celsius:

37.5

Temperature in fahrenheit is:99.5

Temperature converting into integer value:99

Note: Casting to int removes the decimal part, which may lead to loss of precision.

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.

Code;; **package** Enum\_practice;

**import** java.util.Scanner;

**public** **class** enumSwitch {

**enum** Week{***Monday***,***Tuesday***,***Wednesday***,***Thursday***,***Friday***,***Saturday***,***Sunday***}

**public** **static** **void** main(String[] args) {

Scanner scan=**new** Scanner(System.***in***);

System.***out***.println("Enter a day in a week:");

String day=scan.nextLine();

**try** {

Week w=Week.*valueOf*(day.substring(0, 1).toUpperCase() + day.substring(1).toLowerCase());

System.***out***.println("Position in enum (ordinal): " + w.ordinal());

**switch**(w){

**case** ***Monday***:System.***out***.println("It is a Monday");

**break**;

**case** ***Tuesday***:System.***out***.println("It is a Tuesday");

**break**;

**case** ***Wednesday***:System.***out***.println("It is a Wednesday");

**break**;

**case** ***Thursday***:System.***out***.println("It is a Thursday");

**break**;

**case** ***Friday***:System.***out***.println("It is a Friday");

**break**;

**case** ***Saturday***:System.***out***.println("It is a Saturday");

**break**;

**case** ***Sunday***:System.***out***.println("It is a Sunday");

**break**;

}

**if**(w==Week.***Sunday*** || w==Week.***Saturday***) {

System.***out***.println("It is a weekend ");

}

**else** {

System.***out***.println("It is a weekday");

}

}**catch** (IllegalArgumentException e) {

System.***out***.println("Invalid day entered. Please enter a valid day like 'Monday'.");

} **finally** {

scan.close();

}

}

}

Output; Enter a day in a week:

thursday

Position in enum (ordinal): 3

It is a Thursday

It is a weekday

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.

Code; **package** Enum\_practice;

**import** java.util.\*;

**public** **class** EnumDirections {

**enum** directions{***NORTH***,***SOUTH***,***EAST***,***WEST***}

**public** **static** **void** main(String[] args) {

**try** {

Scanner scan= **new** Scanner(System.***in***);

System.***out***.println("Enter any direction name:");

String s=scan.nextLine().toUpperCase();

directions n=directions.*valueOf*(s);

**switch**(n) {

**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**;

**default**:

scan.close();

}

}**catch**(IllegalArgumentException e) {

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

}

}

}

**Output;** Enter any direction name:

north

Move north

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.  
code; **package** Enum\_practice;

**import** java.util.\*;

**public** **class** enumShape {

**enum** Shape{

***CIRCLE*** {

@Override

**public** **double** area(**double**... params) {

**double** radius = params[0];

**return** Math.***PI*** \* radius \* radius;

}

},

***SQUARE***{

@Override

**public** **double** area(**double**... params) {

**double** side = params[0];

**return** side \* side;

}

},

***RECTANGLE***{

@Override

**public** **double** area(**double**... params) {

**double** length = params[0];

**double** breadth = params[1];

**return** length \* breadth;

}

},

***TRIANGLE***{

@Override

**public** **double** area(**double**... params) {

**double** base = params[0];

**double** height = params[1];

**return** 0.5 \* base \* height;

}

};

**public** **abstract** **double** area(**double**... params) ;

}

**public** **static** **void** main(String[] args) {

**double** radius = 5.0;

**double** side = 4.0;

**double** length = 6.0, breadth = 3.0;

**double** base = 8.0, height = 4.0;

// Loop through all shapes and compute area

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

**double** result = **switch** (shape) {

**case** ***CIRCLE*** -> shape.area(radius);

**case** ***SQUARE*** -> shape.area(side);

**case** ***RECTANGLE*** -> shape.area(length, breadth);

**case** ***TRIANGLE*** -> shape.area(base, height);

};

System.***out***.printf("Area of %s: %.2f%n", shape.name(), result);

}

}

}

Output; Area of CIRCLE: 78.54

Area of SQUARE: 16.00

Area of RECTANGLE: 18.00

Area of TRIANGLE: 16.00

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.

Code; **package** Enum\_practice;

**import** java.util.\*;

**enum** Suit{***CLUBS***,***DIAMONDS***,***HEART***,***SPADES***}

**enum** Rank{***ACE***,***TWO***,***THREE***,***FOUR***,***FIVE***,***SIX***,***SEVEN***,***EIGHT***,***NINE***,***TEN***,***JACK***,***QUEEN***,***KING***}

**class** Card1 {

**private** **final** Suit suit;

**private** **final** Rank rank;

**public** Card1(Suit suit, Rank rank) {

**this**.suit = suit;

**this**.rank = rank;

}

**public** String toString() {

**return** rank + " of " + suit;

}

}

**class** Deck {

**private** **final** List<Card1> cards = **new** ArrayList<>();

**public** Deck() {

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

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

cards.add(**new** Card1(suit, rank));

}

}

}

**public** **void** shuffle() {

Collections.*shuffle*(cards);

}

**public** **void** printDeck() {

**for** (Card1 card : cards) {

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

}

}

}

**public** **class** Card{

**public** **static** **void** main(String[] args) {

Deck deck = **new** Deck();

deck.shuffle();

deck.printDeck();

}

}

Output; SEVEN of SPADES

THREE of DIAMONDS

KING of DIAMONDS

TWO of CLUBS

EIGHT of HEART

FIVE of HEART

QUEEN of DIAMONDS

EIGHT of DIAMONDS

SEVEN of CLUBS

SEVEN of HEART

FIVE of DIAMONDS

KING of CLUBS

QUEEN of HEART

FIVE of SPADES

JACK of HEART

NINE of HEART

KING of HEART

EIGHT of SPADES

TWO of SPADES

FOUR of CLUBS

SIX of DIAMONDS

NINE of CLUBS

NINE of DIAMONDS

QUEEN of CLUBS

TEN of DIAMONDS

FOUR of SPADES

JACK of SPADES

SIX of CLUBS

TWO of HEART

TWO of DIAMONDS

QUEEN of SPADES

FOUR of HEART

ACE of SPADES

ACE of DIAMONDS

JACK of DIAMONDS

SIX of HEART

TEN of HEART

ACE of HEART

THREE of CLUBS

THREE of HEART

ACE of CLUBS

NINE of SPADES

EIGHT of CLUBS

SIX of SPADES

FIVE of CLUBS

KING of SPADES

JACK of CLUBS

SEVEN of DIAMONDS

FOUR of DIAMONDS

THREE of SPADES

TEN of SPADES

TEN of CLUBS

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.

Code; **package** Enum\_practice;

**public** **enum** PriorityLevel {

***LOW***(1),

***MEDIUM***(3),

***HIGH***(7),

***CRITICAL***(10);

**private** **final** **int** severity;

**private** **static** **final** **int** ***URGENCY\_THRESHOLD*** = 5;

PriorityLevel(**int** severity) {

**this**.severity = severity;

}

**public** **int** getSeverity() {

**return** severity;

}

**public** **boolean** isUrgent() {

**return** severity >= ***URGENCY\_THRESHOLD***;

}

**public** String getDescription() {

**return** name() + " (Severity: " + severity + ", Urgent: " + isUrgent() + ")";

}

}

**public** **class** Enum\_example {

**public** **static** **void** main(String[] args) {

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

System.***out***.println(level.getDescription());

}

}

}

Output; LOW (Severity: 1, Urgent: false)

MEDIUM (Severity: 3, Urgent: false)

HIGH (Severity: 7, Urgent: true)

CRITICAL (Severity: 10, Urgent: true)

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.  
code; **package** Enum\_practice;

**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** Traffic\_Lights\_State\_Machine {

**public** **static** **void** main(String[] args) {

State current = TrafficLight.***RED***;

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

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

current = current.next();

}

}

}

Output; Current Light: RED

Current Light: GREEN

Current Light: YELLOW

Current Light: RED

Current Light: GREEN

Current Light: YELLOW

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.

Code; **package** Enum\_practice;

**enum** Difficulty{***EASY***,***MEDIUM***,***HARD***}

**class** Game {

**private** Difficulty diff;

Game(Difficulty diff){

**this**.diff=diff;

**switch**(diff) {

**case** ***EASY***:

System.***out***.println("3000 bullets");

**break**;

**case** ***MEDIUM***:

System.***out***.println("2000 bullets");

**break**;

**case** ***HARD***:

System.***out***.println("1000 bullets");

**break**;

}

}

}

**public** **class** Game\_level {

**public** **static** **void** main(String[] args) {

**new** Game(Difficulty.***EASY***);

**new** Game(Difficulty.***MEDIUM***);

**new** Game(Difficulty.***HARD***);

}

}

Output; 3000 bullets

2000 bullets

1000 bullets

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.

Code; **package** Enum\_practice;

**import** java.util.Scanner;

**enum** Operation{

***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***:

**return** a/b;

**default**:

**throw** **new** UnsupportedOperationException("Unknown operation: " + **this**);

}

}

}

**public** **class** Calculator\_operations {

**public** **static** **void** main(String[] args) {

Scanner scan=**new** Scanner(System.***in***);

System.***out***.println("Enter a value:");

**double** a=scan.nextDouble();

System.***out***.println("Enter b value:");

**double** b=scan.nextDouble();

System.***out***.println("PLUS: " + Operation.***PLUS***.eval(a, b));

System.***out***.println("MINUS: " + Operation.***MINUS***.eval(a, b));

System.***out***.println("TIMES: " + Operation.***TIMES***.eval(a, b));

System.***out***.println("DIVIDE: " + Operation.***DIVIDE***.eval(a, b));

scan.close();

}

}

Output; Enter a value:

10

Enter b value:

5

PLUS: 15.0

MINUS: 5.0

TIMES: 50.0

DIVIDE: 2.0

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

Code; **package** Enum\_practice;

**import** java.util.Scanner;

**enum** Operations {

***PLUS*** {

@Override

**public** **double** eval(**double** a, **double** b) {

**return** a + b;

}

},

***MINUS*** {

@Override

**public** **double** eval(**double** a, **double** b) {

**return** a - b;

}

},

***TIMES*** {

@Override

**public** **double** eval(**double** a, **double** b) {

**return** a \* b;

}

},

***DIVIDE*** {

@Override

**public** **double** eval(**double** a, **double** b) {

**return** a / b;

}

};

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

}

**public** **class** Calulate\_method\_overrides {

**public** **static** **void** main(String[] args) {

Scanner scan = **new** Scanner(System.***in***);

System.***out***.println("Enter a value:");

**double** a = scan.nextDouble();

System.***out***.println("Enter b value:");

**double** b = scan.nextDouble();

System.***out***.println("PLUS: " + Operations.***PLUS***.eval(a, b));

System.***out***.println("MINUS: " + Operations.***MINUS***.eval(a, b));

System.***out***.println("TIMES: " + Operations.***TIMES***.eval(a, b));

System.***out***.println("DIVIDE: " + Operations.***DIVIDE***.eval(a, b));

scan.close();

}

}

Output; Enter a value:

25

Enter b value:

2

PLUS: 27.0

MINUS: 23.0

TIMES: 50.0

DIVIDE: 12.5

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.  
code; **package** Enum\_practice;

**import** java.util.Scanner;

**enum** KnowledgeLevel{

***BEGINNER***,***ADVANCED***,***PROFESSIONAL***,***MASTER***;

**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** Knowledge\_Test {

**public** **static** **void** main(String[] args) {

Scanner scan=**new** Scanner(System.***in***);

System.***out***.println("Enter a score :");

**int** score=scan.nextInt();

**try** {

KnowledgeLevel level = KnowledgeLevel.*fromScore*(score);

System.***out***.println("Your level is: " + level);

} **catch** (IllegalArgumentException e) {

System.***out***.println(e.getMessage());

}

scan.close();

}

}

Output; Enter a score :

5

Your level is: ADVANCED

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.");

}

Code; **package** Exception\_practice;

**public** **class** ExceptionDemo {

**public** **static** **void** main(String[] args) {

**try** {

**int** a=10;

**int** b=0;

**int** c=a/b;

}**catch**(ArithmeticException e) {

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

}

**finally** {

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

}

**try** {

**int**[] arr= {1,0,4,5,6};

**int** a1=arr[5];

}**catch**(ArrayIndexOutOfBoundsException e) {

System.***out***.println("Array index is out of bound is not allowed!");

}

**finally** {

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

}

}

}

Output; Division by zero is not allowed!

Operation completed.

Array index is out of bound is not allowed!

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); }

}

Code; **package** Exception\_practice;

**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**[] testValues = {2, 5, 8, 11};

**for** (**int** value : testValues) {

**try** {

*CheckOdd*(value);

} **catch** (OddNumberException e) {

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

}

}

}

}

Output; 2 is even.

Caught Exception: Odd number: 5

8 is even.

Caught Exception: Odd number: 11

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.

Code; **package** Exception\_practice;

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**public** **class** FileReadDemo {

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

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

String firstLine = reader.readLine();

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

reader.close();

}

**public** **static** **void** main(String[] args) {

String filename = "sample.txt";

**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");

}

}

}

Output; File not found:sample.txt

Cleanup done

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".

Code; **package** Exception\_practice;

**import** java.io.\*;

**public** **class** MultiExceptionDemo {

**public** **static** **void** main(String[] args) {

**try** {

BufferedReader reader = **new** BufferedReader(**new** FileReader("input.txt"));

String line = reader.readLine();

**int** number = Integer.*parseInt*(line);

**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.");

}

}

}

Output; File not found

Execution completed.