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.

ANSWER

```
import java.util.Scanner;


public class TypeConversionDemo {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);


      // Implicit widening
      System.out.print("Enter an integer: ");
      int intVal = sc.nextInt();
      double widened = intVal; // implicit widening
      System.out.println("Integer: " + intVal + ", Double (widened): " + widened);


      // Explicit narrowing
      System.out.print("Enter a double: ");
      double d = sc.nextDouble();
      int intCast = (int) d;   // explicit narrowing
      short shortCast = (short) d; // further narrowing
      System.out.println("Original double: " + d);
      System.out.println("Cast to int: " + intCast);
      System.out.println("Cast to short: " + shortCast); // may overflow


      // Int <-> String conversion
      int num = 123;
```

```
        String str = String.valueOf(num);
        System.out.println("Converted int to String: " + str);

        try {
            int parsed = Integer.parseInt(str);
            System.out.println("Parsed back to int: " + parsed);
        } catch (NumberFormatException e) {
            System.out.println("Invalid number format: " + e.getMessage());
        }

        sc.close();
    }
}
```

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?
```

ANSWER

```
public class CompoundAssignmentDemo {
    public static void main(String[] args) {
        int x = 5;
        x += 4.5;
        System.out.println("After x += 4.5, x = " + x)  }
```

}

3. 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();
```

ANSWER

```
class Animal {
  public void makeSound() {
    System.out.println("Some generic sound...");
  }
}

class Dog extends Animal {
  @Override
  public void makeSound() {
    System.out.println("Woof!");
  }
  public 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();
      // a.fetch();
   }
}
```

4) Mini-Project – Temperature Converter

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

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

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

ANSWER

```
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 truncated = (int) fahrenheit;

        System.out.println("Fahrenheit (precise): " + fahrenheit);
        System.out.println("Fahrenheit (truncated to int): " + truncated);
        sc.close();
    }
}
```

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.

ANSWER

```
import java.util.Scanner;

enum DaysOfWeek {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}

public class DaysOfWeekDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a day of the week: ");
        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("It's a weekend!");
                    break;
                default:
                    System.out.println("It's a weekday.");
            }
        } catch (IllegalArgumentException e) {
            System.out.println("Invalid day entered!");
        }
        sc.close();
    }
}
```

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.

ANSWER

```
import java.util.Scanner;


enum Direction { NORTH, SOUTH, EAST, WEST }
```

```
public class DirectionDemo {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter 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");
            case SOUTH -> System.out.println("Move south");
            case EAST -> System.out.println("Move east");
            case WEST -> System.out.println("Move west");
         }
      } catch (IllegalArgumentException e) {
         System.out.println("Invalid direction entered!");
      }
      sc.close();
   }
}
```

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.

ANSWER

```
enum Shape {
   CIRCLE {
```

```
    @Override double area(double... params) { return Math.PI * params[0] * params[0]; }
  },
  SQUARE {
    @Override double area(double... params) { return params[0] * params[0]; }
  },
  RECTANGLE {
    @Override double area(double... params) { return params[0] * params[1]; }
  },
  TRIANGLE {
    @Override double area(double... params) { return 0.5 * params[0] * params[1]; }
  };

  abstract double area(double... params);
}

public class ShapeDemo {
  public static void main(String[] args) {
    System.out.println("Circle area (r=5): " + Shape.CIRCLE.area(5));
    System.out.println("Square area (a=4): " + Shape.SQUARE.area(4));
    System.out.println("Rectangle area (w=4,h=6): " + Shape.RECTANGLE.area(4, 6));
    System.out.println("Triangle area (b=5,h=3): " + Shape.TRIANGLE.area(5, 3));
  }
}
```

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.

ANSWER

```
import java.util.*;

enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }
enum Rank { ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING }

class Card {
   private final Suit suit;
   private final Rank rank;
   public Card(Suit suit, Rank rank) { this.suit = suit; this.rank = rank; }
   @Override public String toString() { return rank + " of " + suit; }
}

class Deck {
   private List<Card> cards = new ArrayList<>();
   public Deck() {
      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() { cards.forEach(System.out::println); }
}

public class CardGame {
   public static void main(String[] args) {
```

```
    Deck deck = new Deck();
    deck.shuffle();
    deck.printDeck();
  }
}
```

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.

ANSWER

```
enum PriorityLevel {
  LOW(1), MEDIUM(2), HIGH(3), CRITICAL(4);

  private final int severity;
  PriorityLevel(int severity) { this.severity = severity; }
  public boolean isUrgent() { return severity >= 3; }
  public int getSeverity() { return severity; }
}

public class PriorityDemo {
  public static void main(String[] args) {
    for (PriorityLevel p : PriorityLevel.values()) {
      System.out.println(p + " (severity=" + p.getSeverity() + ") urgent? " + p.isUrgent());
    }
  }
}
```

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.

ANSWER

```
interface State { State next(); }


enum TrafficLight implements State {
   RED { public State next() { return GREEN; } },
   GREEN { public State next() { return YELLOW; } },
   YELLOW { public State next() { return RED; } };
}


public class TrafficLightDemo {
   public static void main(String[] args) {
      State current = TrafficLight.RED;
      for (int i = 0; i < 6; i++) {
         System.out.println("Current: " + current);
         current = current.next();
      }
   }
}
```

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.

ANSWER

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


class Game {
   private int bullets;
   public Game(Difficulty diff) {
      switch (diff) {
         case EASY -> bullets = 3000;
         case MEDIUM -> bullets = 2000;
         case HARD -> bullets = 1000;
      }
      System.out.println("Game started with " + bullets + " bullets.");
   }
}


public class GameDemo {
   public static void main(String[] args) {
      new Game(Difficulty.EASY);
      new Game(Difficulty.MEDIUM);
      new Game(Difficulty.HARD);
   }
}
```

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.

ANSWER

```
enum OperationSwitch {
   PLUS, MINUS, TIMES, DIVIDE;
   public double eval(double a, double b) {
      return switch (this) {
         case PLUS -> a + b;
         case MINUS -> a - b;
         case TIMES -> a * b;
         case DIVIDE -> a / b;
      };
   }
}

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) { return a / b; } };

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

public class OperationDemo {
   public static void main(String[] args) {
      System.out.println("Switch-based: 10 + 5 = " + OperationSwitch.PLUS.eval(10, 5));
      System.out.println("Override-based: 10 / 2 = " + OperationOverride.DIVIDE.eval(10, 2));
```

```
  }
}
```

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.

ANSWER

```
enum KnowledgeLevel {
  BEGINNER, ADVANCED, PROFESSIONAL, MASTER;

  public static KnowledgeLevel fromScore(int score) {
    if (score >= 0 && score <= 3) return BEGINNER;
    else if (score <= 6) return ADVANCED;
    else if (score <= 9) return PROFESSIONAL;
    else if (score == 10) return MASTER;
    else throw new IllegalArgumentException("Score out of range!");
  }
}

public class KnowledgeDemo {
  public static void main(String[] args) {
    int[] scores = {0, 3, 4, 6, 7, 9, 10};
    for (int s : scores) {
      System.out.println("Score: " + s + " → " + KnowledgeLevel.fromScore(s));
    }
  }
}
```

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.");
}
```

ANSWER

```
public class ExceptionDemo {
  public static void main(String[] args) {
    // Division by zero
    try {
      int a = 10 / 0;
    } catch (ArithmeticException e) {
      System.out.println("Division by zero is not allowed!");
    } finally {
      System.out.println("Operation completed.");
    }
```

```
        // Array access out of bounds
        try {
            int[] arr = {1, 2, 3};
            int val = arr[5];
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index out of bounds!");
        } 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 { /* ... */ }
```

2. If n is odd, throw a custom checked exception OddNumberException with message "Odd number: " + n.
3. 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); }
}
```

ANSWER

```
// Custom Exception
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};

    for (int n : numbers) {
      try {
        checkOdd(n);
      } catch (OddNumberException e) {
        System.out.println("Caught Exception → " + e.getMessage());
      }
    }
  }
}
```

3: 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.

ANSWER

```
import java.io.*;


public class FileReadDemo {
  public static void readFile(String filename) throws FileNotFoundException, IOException {
    BufferedReader br = new BufferedReader(new FileReader(filename));
    String line = br.readLine();
    System.out.println("First line: " + line);
    br.close();
  }


  public static void main(String[] args) {
    String file = "test.txt"; // change to your file


    try {
      readFile(file);
    } catch (FileNotFoundException e) {
      System.out.println("File not found: " + file);
    } catch (IOException e) {
      System.out.println("Error reading file: " + e.getMessage());
    } finally {
      System.out.println("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:

0. FileNotFoundException
1. IOException
2. NumberFormatException
3. ArithmeticException

- In each catch, print a tailored message:
  - File not found
  - Problem reading file
  - Invalid number format
  - Division by zero
- Finally, print "Execution completed".

ANSWER

```
import java.io.*;

public class FileReadDemo {
  public static void readFile(String filename) throws FileNotFoundException, IOException {
    BufferedReader br = new BufferedReader(new FileReader(filename));
    String line = br.readLine();
```

```
        System.out.println("First line: " + line);
        br.close();
    }

    public static void main(String[] args) {
        String file = "test.txt"; // change to your file

        try {
            readFile(file);
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + file);
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        } finally {
            System.out.println("Cleanup done.");
        }
    }
}
```