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.

```
package day6_Assessment;
import java.util.Scanner;

public class TypeCasting {
   public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       System.out.print("Enter an integer: ");
       int intValue = sc.nextInt();
       double widenedValue = intValue;
       System.out.println("Integer value: " + intValue);
       System.out.println("After implicit widening to double: " + widenedValue);
       System.out.print("Enter a double: ");
       double doubleValue = sc.nextDouble();
       int castToInt = (int) doubleValue;
       short castToShort = (short) doubleValue;
       System.out.println("Original double: " + doubleValue);
       System.out.println("After explicit cast to int: " + castToInt);
       System.out.println("After explicit cast to short: " + castToShort);
       sc.close();
   }
}
```

Output:

```
Enter an integer: 25
Integer value: 25
After implicit widening to double: 25.0
Enter a double: 10.456
Original double: 10.456
After explicit cast to int: 10
After explicit cast to short: 10
```

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

```
package day6_Assessment;
public class IntStringConversion {
   public static void main(String[] args) {
      int num = 123;
      String str = String.valueOf(num);
      System.out.println("Integer to String: " + str);
      try {
         int parsedNum = Integer.parseInt(str);
         System.out.println("String back to Integer: " + parsedNum);
      } catch (NumberFormatException e) {
         System.out.println("Invalid number format: " + e.getMessage());
      }
   }
}
```

Output:

```
Integer to String: 123
String back to Integer: 123
```

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?

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

```
package day6_Assessment;
public class CompoundAssignment {
    public static void main(String[] args) {
        int x = 5;
        // x = x + 4.5;
        //Compile-time error: possible lossy conversion from
double to int
```

```
        x += 4.5;
        //Works: implicit narrowing conversion
        System.out.println("Value of x after compound assignment: " + x);
    }
}
```

Output:

```
Value of x after compound assignment: 9
```

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();
```

```java
package day6_Assessment;
class Animal {
    void makeSound() {
        System.out.println("Some animal sound");
    }
}
class Dog extends Animal {
    void makeSound() {
        System.out.println("Woof!");
    }
    void fetch() {
        System.out.println("Dog is fetching the ball!");
    }
}
public class ObjectCasting {
    public static void main(String[] args) {
        Dog d = new Dog();
        Animal a = d;
        a.makeSound();
    }
}
```

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;

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

```
package day6_Assessment;
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 truncatedFahrenheit = (int) fahrenheit;
        System.out.println("Fahrenheit (precise): " + fahrenheit);
        System.out.println("Fahrenheit (truncated to int): " + truncatedFahrenheit);
        System.out.println("Note: Truncating removes decimal part, causing precision loss.");
        sc.close();
    }
}
```

Output:

```
Enter temperature in Celsius: 48.26
Fahrenheit (precise): 118.868
Fahrenheit (truncated to int): 118
```

Note: Truncating removes decimal part, causing 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.

```
package day6_Assessment;
import java.util.Scanner;

public class DayaOfWeek {
   enum DaysOfWeeks {
      MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
   }

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

      System.out.print("Enter a day of the week: ");
      String input = scanner.nextLine().trim().toUpperCase();

      try {
         DaysOfWeeks day = DaysOfWeeks.valueOf(input);
         System.out.println("Position of " + day + " is: " + day.ordinal());

         switch (day) {
            case SATURDAY:
            case SUNDAY:
               System.out.println(day + " is a weekend!");
               break;
            default:
               System.out.println(day + " is a weekday.");
               break;
         }

      } catch (IllegalArgumentException e) {
         System.out.println("Invalid day! Please enter a valid day of the week.");
      }

      scanner.close();
   }
}
```

Output:

```
Enter a day of the week: Monday
Position of MONDAY is: 0
MONDAY 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.

```
package day6_Assessment;
import java.util.Scanner;
public class CompassDirections {
    enum Direction {
        NORTH, SOUTH, EAST, WEST
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter direction (NORTH, SOUTH, EAST, WEST): ");
        String input = scanner.nextLine().trim().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.");
        }
```

```java
        scanner.close();
    }
}
```

Output:

```
Enter direction (NORTH, SOUTH, EAST, WEST): 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.

```java
package day6_Assessment;
public class ShapeAreaCalculator {
    public static void main(String[] args) {
        for (Shape shape : Shape.values()) {
            double result = 0.0;
            switch (shape) {
                case CIRCLE:
                    result = shape.area(5);
                    break;
                case SQUARE:
                    result = shape.area(4);
                    break;
                case RECTANGLE:
                    result = shape.area(4, 6);
                    break;
                case TRIANGLE:
                    result = shape.area(3, 5);
                    break;
            }
            System.out.println(shape + " area: " + result);
        }
    }
}
enum Shape {
    CIRCLE {
        public double area(double... params) {
            double radius = params[0];
            return Math.PI * radius * radius;
        }
    },
    SQUARE {
        public double area(double... params) {
            double side = params[0];
            return side * side;
        }
    },
    RECTANGLE {
        public double area(double... params) {
            double width = params[0];
            double height = params[1];
            return width * height;
        }
    },
    TRIANGLE {
        public double area(double... params) {
            double base = params[0];
            double height = params[1];
            return 0.5 * base * height;
        }
    };
    public abstract double area(double... params);
}
```

Output:

```
CIRCLE area: 78.53981633974483
SQUARE area: 16.0
RECTANGLE area: 24.0
TRIANGLE area: 7.5
```

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.

Program:

=========

```
package day6_Assessment;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class CardGame {
    public static void main(String[] args) {
        Deck deck = new Deck();
        deck.shuffle();
        deck.printDeck();
    }
}
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;
    }
    public String toString() {
        return rank + " of " + suit;
    }
}
class Deck {
    private final 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() {
        for (Card card : cards) {
            System.out.println(card);
        }
    }
}
```

Output:

```
TEN of SPADES
SIX of CLUBS
SIX of DIAMONDS
TEN of HEARTS
NINE of CLUBS
FOUR of DIAMONDS
QUEEN of SPADES
KING of DIAMONDS
EIGHT of CLUBS
TWO of DIAMONDS
JACK of CLUBS
KING of CLUBS
FIVE of HEARTS
EIGHT of HEARTS
SEVEN of SPADES
ACE of CLUBS
QUEEN of HEARTS
KING of HEARTS
THREE of CLUBS
SIX of HEARTS
NINE of HEARTS
ACE of DIAMONDS
```

TWO of SPADES

JACK of DIAMONDS

SIX of SPADES

FIVE of CLUBS

JACK of HEARTS

QUEEN of DIAMONDS

NINE of DIAMONDS

FOUR of CLUBS

THREE of DIAMONDS

TWO of CLUBS

SEVEN of DIAMONDS

FIVE of SPADES

SEVEN of HEARTS

TEN of CLUBS

TWO of HEARTS

KING of SPADES

EIGHT of DIAMONDS

NINE of SPADES

QUEEN of CLUBS

EIGHT of SPADES

TEN of DIAMONDS

JACK of SPADES

FIVE of DIAMONDS

```
ACE of HEARTS
THREE of HEARTS
SEVEN of CLUBS
ACE of SPADES
FOUR of HEARTS
FOUR of SPADES
THREE of SPADES
```

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.

```java
package day6_Assessment;
public class PriorityDemo {
   public static void main(String[] args) {
      for (PriorityLevel level : PriorityLevel.values()) {
         System.out.println("Priority: " + level);
         System.out.println("Severity Code: " + level.getSeverity());
         System.out.println("Is Urgent? " + level.isUrgent());
         System.out.println("--------------------------");
      }
   }
}
enum PriorityLevel {
   LOW(1),
   MEDIUM(3),
   HIGH(5),
   CRITICAL(8);
   private final int severity;
   PriorityLevel(int severity) {
      this.severity = severity;
   }
   public int getSeverity() {
      return severity;
   }
   public boolean isUrgent() {
      return severity >= 5;
   }
}
```

Output:

```
Priority: LOW
Severity Code: 1
Is Urgent? false
---------------------------
Priority: MEDIUM
Severity Code: 3
Is Urgent? false
---------------------------
Priority: HIGH
Severity Code: 5
Is Urgent? true
---------------------------
Priority: CRITICAL
Severity Code: 8
Is 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.

```
package day6_Assessment;
public class TrafficLightDemo {
    public static void main(String[] args) {
        TrafficLight light = TrafficLight.RED;
        for (int i = 0; i < 6; i++) {
            System.out.println("Current light: " + light);
            light = (TrafficLight) light.next();
        }
    }
}
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;
        }
    };
}
```

```
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.

```
package day6_Assessment;
public class GameDemo {
    public static void main(String[] args) {
        Game easyGame = new Game(Difficulty.EASY);
        Game mediumGame = new Game(Difficulty.MEDIUM);
        Game hardGame = new Game(Difficulty.HARD);
    }
```

```java
}

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:
                bullets = 0;
        }
        System.out.println("Game started with difficulty: " + diff + ", Bullets: " + bullets);
    }
}
```

Output:
```
Game started with difficulty: EASY, Bullets: 3000
Game started with difficulty: MEDIUM, Bullets: 2000
Game started with difficulty: HARD, Bullets: 1000
```

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.

```java
package day6_Assessment;
public class CalculatorDemo {
   public static void main(String[] args) {
      double a = 10, b = 2;
      System.out.println("Calculator using switch-based enum:");
      for (Operation op : Operation.values()) {
         try {
            System.out.println(op + ": " + op.eval(a, b));
         } catch (ArithmeticException e) {
            System.out.println(op + ": Error - " + e.getMessage());
         }
      }
   }
   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:
               if (b == 0) throw new ArithmeticException("Cannot divide by zero");
               return a / b;
            default:
               throw new AssertionError("Unknown operation " + this);
         }
      }
   }
}
```

Output:

```
Calculator using switch-based enum:
PLUS: 12.0
MINUS: 8.0
TIMES: 20.0
DIVIDE: 5.0
```

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.

```java
package day6_Assessment;
public class KnowledgeLevelDemo {
   public static void main(String[] args) {
      int[] testScores = {0, 3, 4, 6, 7, 9, 10, -1, 11};
      for (int score : testScores) {
         try {
            KnowledgeLevel level = KnowledgeLevel.fromScore(score);
            System.out.println("Score: " + score + " → Level: " + level);
         } catch (IllegalArgumentException e) {
            System.out.println("Score: " + score + " → " + e.getMessage());
         }
      }
   }
   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);
         }
      }
   }
}
```

Output:

```
Score: 0 → Level: BEGINNER
Score: 3 → Level: BEGINNER
Score: 4 → Level: ADVANCED
Score: 6 → Level: ADVANCED
Score: 7 → Level: PROFESSIONAL
Score: 9 → Level: PROFESSIONAL
Score: 10 → Level: MASTER
Score: -1 → Invalid score: -1
Score: 11 → Invalid score: 11
```

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.");
}
```

```
package day6_Assessment;
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};
            int value = numbers[5];
            System.out.println("Value: " + value);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index is out of bounds!");
        } finally {
            System.out.println("Operation completed.");
        }
    }
}
```

Output:

```
Division by zero is not allowed!
Operation completed.

Array index is out of bounds!
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); }

```
}
```

```java
package day6_Assessment;
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, 7, 4, 9, 6};

      for (int num : numbers) {
         try {
            checkOdd(num);
         } catch (OddNumberException e) {
            System.out.println("Exception caught: " + e.getMessage());
         }
      }
   }
   public static class OddNumberException extends Exception {
      public OddNumberException(String message) {
         super(message);
      }
   }
}
```

Output:

```
2 is even.
Exception caught: Odd number: 7
4 is even.
Exception caught: Odd number: 9
6 is even.
```

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()"

Include a finally block that prints "Cleanup done."

4. regardless of outcome.

```
 package day6_Assessment;
import java.io.*;
public class MultipleCatches {
       public static void main(String[] args) {
    String filename = "input.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.");
    }
  }
  public static void readFile(String filename) throws FileNotFoundException, IOException {
    BufferedReader reader = new BufferedReader(new FileReader(filename));
    String line = reader.readLine();
    System.out.println("First line: " + line);
    reader.close();
  }
}
```

Output:

```
File not found: input.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:

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

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

```
package day6_Assessment;
import java.io.*;
public class MultiExceptionDemo {
		public static void main(String[] args) {
        BufferedReader reader = null;
        try {
            File file = new File("input.txt");
            reader = new BufferedReader(new FileReader(file));
            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");
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException e) {
            }
        }
    }
}
```

Output:

```
File not found
Execution completed
```