A INTERNSHIP PROJECT ON

DEVELOPING A JAVA BASED TIC TAC TOE GAME AND NUMBER GUESSING GAME



An Internship Project carried out of six phrase, submitted in partial fulfillment of the requirements for the award of the degree of

**BACHELOR OF TECHNOLOGY IN**

**COMPUTER SCIENCE AND BUSINESS SYSTEMS**

**BY**

NAME : GUNAMPALLI MEGHANA REDDY REGD.NO: 229X1A2908

Department of Emerging Technologies In Computer Science

**G. Pulla Reddy Engineering College (Autonomous):**

**Kurnool–518007, Andhra Pradesh, India**

**(Affiliated to Jawaharlal Nehru Technological University – Anantapur)**

**Internship Project carried out at: Six Phrase, Chennai Internship Project Report**

**Submitted in accordance with the requirement for the degree of BACHELOR OF TECHNOLOGY**

Name of the College : G. PULLA REDDY ENGINEERING COLLEGE

Department : EMERGING TECHNOLOGIES IN COMPUTER SCIENCE

Name of the Mentor : 1. Gayathri

2. Jannath Sadiq

Company Name : Six Phrase, Chennai Title of the Project :

1. TIC TAC TOE GAME
2. NUMBER GUESSING GAME

Name of the Student: GUNAMPALLI MEGHANA REDDY

Regd Number : 229X1A2908 Program of study : Summer Internship

Date of Submission : 28-05-2025

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to my internal guide, Ms .Gayathri / Jannath Sadiq and to the organization Six Phrase for giving me this opportunity to work on the project titled Tic Tac Toe Game and Number Guessing Game. I would also like to thank my classmates and friends for their constant support and encouragement throughout this internship.

# Table of Contents

|  |  |  |
| --- | --- | --- |
| **S. No** | **Contents** | **Page No.** |
| **1.** | **TIC TAC TOE GAME** | 5 |
|  | 1.1 Abstract | 5 |
|  | 1.2 Introduction | 5 |
|  | 1.3 Objective of the Project | 5 |
|  | 1.4 Functional Requirements | 5-6 |
|  | 1.5 Technologies Used | 6 |
|  | 1.6 Project Structure | 6-7 |
|  | 1.7 Implementation Details | 8-11 |
|  | 1.8 Output | 12 |
|  | 1.9 Exception Handling | 13 |
|  | 1.10 Conclusion | 13 |
| **2.** | **NUMBER GUESSING GAME** | 14 |
|  | 2.1 Abstract | 14 |
|  | 2.2 Introduction | 14 |
|  | 2.3 Objective of the Project | 14 |
|  | 2.4 Functional Requirements | 14-15 |
|  | 2.5 Technologies Used | 15 |
|  | 2.6 Project Structure | 15-16 |
|  | 2.7 Implementation Details | 16-19 |
|  | 2.8 Output | 19 |
|  | 2.9 Exception Handling | 20 |
|  | 2.10 Conclusion | 20 |
| **3.** | **Outcomes** | 21 |
| **4.** | **References** | 21 |

**JavaProject1: Tic Tac Toe Game**

* 1. **Abstract**

This project implements a console-based two-player Tic Tac Toe game using Java. The game allows two users to take turns, enter their moves, and receive visual feedback via a 3x3 grid. It handles invalid inputs, detects winners or draws, and demonstrates the use of Java classes, exception handling, and modular programming using packages.

# Introduction

Tic Tac Toe is a classic paper-and-pencil game played between two players who alternate marking spaces in a 3x3 grid with X and O. The first player to get three of their marks in a horizontal, vertical, or diagonal row wins the game. This project recreates the game logic in Java and runs it via the console, using an object-oriented approach with modular code design.

# Objective of the Project

To develop a user-friendly Java console application that:

* + - Simulates a two-player Tic Tac Toe game using a 3x3 grid.
    - Applies object-oriented programming principles like encapsulation and modularity.
    - Validates user input and prevents invalid or duplicate moves.
    - Implements custom exception handling for robust error management.
    - Determines and announces the game result: win or draw.

# Functional Requirements

## Input Handling:

* + - Accept player names and their symbol choices(X or O).
    - Prompt players to enter row and column values for each move.
    - Validate inputs to ensure values are within the 3x3 grid and cells are not already occupied.
    - Handle invalid input using custom exceptions.

## Game Logic:

* + - Manage player turns, update the board with each move, and check for win or draw conditions after every valid move using structured control flow and logical checks.

## Exception Handling:

* + - InvalidMoveException handles illegal or duplicate cell selections.
    - GameAlreadyOverException prevents further moves once the game is concluded.

## Data Management:

* + - Maintain board state using a 2D character array.
    - Store player details using instances of the Player class.
    - Track game status(ongoing, won or draw) within the service layer.

## Interface:

* + - Provide a console-based interface that displays the board, prompts user input, and shows game results clearly.

# Technologies Used

* + - Programming Language: Java
    - OOP Concepts: Encapsulation, Abstraction, Modular Design
    - Data Structures: 2D Array (for board representation)
    - Exception Handling: Custom Exceptions (InvalidMoveException, GameAlreadyOverException)

# Project Structure

## Class: Player

**Purpose**: Represents a player in the game with their name and symbol.

## Key Components:

* + Name : A private field storing the player’s name.
  + Symbol : Aprivate field storing the player’s chosen symbol (X or O).
  + Constructor: Initializes player name and symbol.
  + Getters: Methods to retrieve player name and symbol.

## Class: Board

**Purpose**: Maintains the 3x3 game grid and supports board operations.

## KeyFeatures:

* + Grid : A 2D character array representing the game board.
  + Constructor: Initializes the board with empty spaces.
  + Methods: display(), isCellEmpty(), placeSymbol(), isFull(), and checkWin() manage board display, validate moves, place symbols, check for board fullness, and determine win conditions.

1. **Custom Exceptions: InvalidMoveException, GameAlreadyOverException Purpose**: Handle invalid moves and actions after game completion.

## Key Features:

* + Extend Java’s Exception class.
  + Accept custom error messages through constructors.
  + Used to interrupt and guide the game flow when errors occur.

## Class: GameService

**Purpose**: Coordinates the game logic, such as turn management and win detection.

## KeyComponents:

* + **Fields**: References to Player objects, Board, current turn, and game status.
  + **Constructor**: Accepts players and initializes a new game board.

## Methods:

* + - makeMove(): Validates and executes a player’s move.
    - checkWinner(): Checks if the current player has won.
    - isDraw(): Determines if the game ends in a draw.
    - isGameOver(): Returns game status.
    - getCurrentPlayer(): Retrieves the player whose turn it is.

## Main Application: TicTacToeApp

**Purpose**: Provides the console-based interface and manages game flow.

## Key Components:

* + **Main Method**:
    - Accepts player input (names).
    - Starts the game loop.
    - Accepts player moves.
    - Displays the board after every move.
    - Announces winner or draw.
    - Offers the option to restart the game.
  + Uses Scanner for input collection and exception handling for smooth gameplay.

**Key Features sand Functionality**

## Turn-Based Gameplay:

Alternates between two players and updates the board after each valid move.

## Win and Draw Detection:

Automatically checks for win or draw conditions after every move.

## Custom Exceptions:

Ensures robust validation for illegal moves and game-over situations.

## Modular OOP Design:

Uses separate classes for players, game board, logic, and exceptions to ensure maintainability and reusability.

## Console-Based Interface:

Clean and interactive text-based interface to guide users through each step of the game.

# Implementation Details

## Player.java

package tic\_tac\_toe.models; public class Player {

private String name; private char symbol; public Player(String name,

char symbol) {

this.name = name; this.symbol = symbol;

}

public String getName() { return name;

}

public char getSymbol() { return symbol;

}

}

## Board.java

package tic\_tac\_toe.models; public class Board {

private char[][] grid; public Board() {

grid = new char[3][3]; for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++) grid[i][j] = ' ';

}

public void display() {

for (int i = 0; i < 3; i++) {

System.out.print(grid[i][0]+ "|"+ grid[i][1] + "|" + grid[i][2]); if (i < 2)

System.out.println("\n ");

}

System.out.println();

}

public boolean isCellEmpty(int row, int col) { return grid[row][col] == ' ';

}

public void placeSymbol(int row, int col, char symbol) { grid[row][col] = symbol;

}

public boolean isFull() { for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++) if (grid[i][j] == ' ')

return false; return true;

}

public boolean checkWin(char symbol) { for (int i = 0; i < 3; i++)

if ((grid[i][0] == symbol && grid[i][1] == symbol && grid[i][2] == symbol) || (grid[0][i] == symbol && grid[1][i] == symbol && grid[2][i] == symbol)) return true;

return (grid[0][0] == symbol && grid[1][1] == symbol && grid[2][2] == symbol) || (grid[0][2] == symbol && grid[1][1] == symbol && grid[2][0] == symbol);

}

}

## InvalidMoveException.java

package tic\_tac\_toe.exceptions;

public class InvalidMoveException extends Exception { public InvalidMoveException(String message) {

super(message);

}

}

## GameAlreadyOverException.java

package tic\_tac\_toe.exceptions;

public class GameAlreadyOverException extends Exception { public GameAlreadyOverException(String message) {

super(message);

}

}

## GameService.java

package tic\_tac\_toe.services; import tic\_tac\_toe.models.Board; import tic\_tac\_toe.models.Player; import tic\_tac\_toe.exceptions.\*; public class GameService {

private Board board;

private Player player1, player2, currentPlayer; private boolean gameOver;

public GameService(Player p1, Player p2) { board = new Board();

this.player1 = p1; this.player2 = p2;

this.currentPlayer = p1; this.gameOver = false;

}

public void playMove(int row, int col) throws InvalidMoveException, GameAlreadyOverException {

if (gameOver)

throw new GameAlreadyOverException("Game is already over!");

if (row < 0 || row >= 3 || col < 0 || col >= 3 || !board.isCellEmpty(row, col)) throw new InvalidMoveException("Invalid move. Try again.");

board.placeSymbol(row, col, currentPlayer.getSymbol()); board.display();

if (board.checkWin(currentPlayer.getSymbol())) { System.out.println("Player " + currentPlayer.getName() + " wins!"); gameOver = true;

} else if (board.isFull()) { System.out.println("It's a draw!"); gameOver = true;

} else {

switchPlayer();

}

}

private void switchPlayer() {

currentPlayer = (currentPlayer == player1) ? player2 : player1;

}

public Player getCurrentPlayer() { return currentPlayer;

}

public boolean isGameOver() { return gameOver;

}

public void restart() { this.board = new Board();

this.currentPlayer = player1; this.gameOver = false;

}

}

**TicTacToeApp.java** package tic\_tac\_toe.main; import java.util.Scanner;

import tic\_tac\_toe.models.Player;

import tic\_tac\_toe.services.GameService; import tic\_tac\_toe.exceptions.\*;

public class TicTacToeApp {

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

System.out.println("--- Tic Tac Toe Game ---"); System.out.print("Enter Player 1 Name: "); String name1 = scanner.nextLine(); System.out.print("Enter Player 2 Name: "); String name2 = scanner.nextLine();

Player p1 = new Player(name1, 'X'); Player p2 = new Player(name2, 'O');

GameService game = new GameService(p1, p2); while (true) {

try { System.out.println(game.getCurrentPlayer().getName()+

" (" + game.getCurrentPlayer().getSymbol() + "), enter your move (row[0-2] and column[0-2]): ");

int row = scanner.nextInt(); int col = scanner.nextInt(); game.playMove(row, col);

}

catch (InvalidMoveException | GameAlreadyOverException e) { System.out.println("Error: " + e.getMessage());

}

if (game.isGameOver()) {

System.out.print("Do you want to play again? (y/n): "); char choice = scanner.next().charAt(0);

if (choice == 'y' || choice == 'Y') { game.restart();

} else {

break;

}

}

}

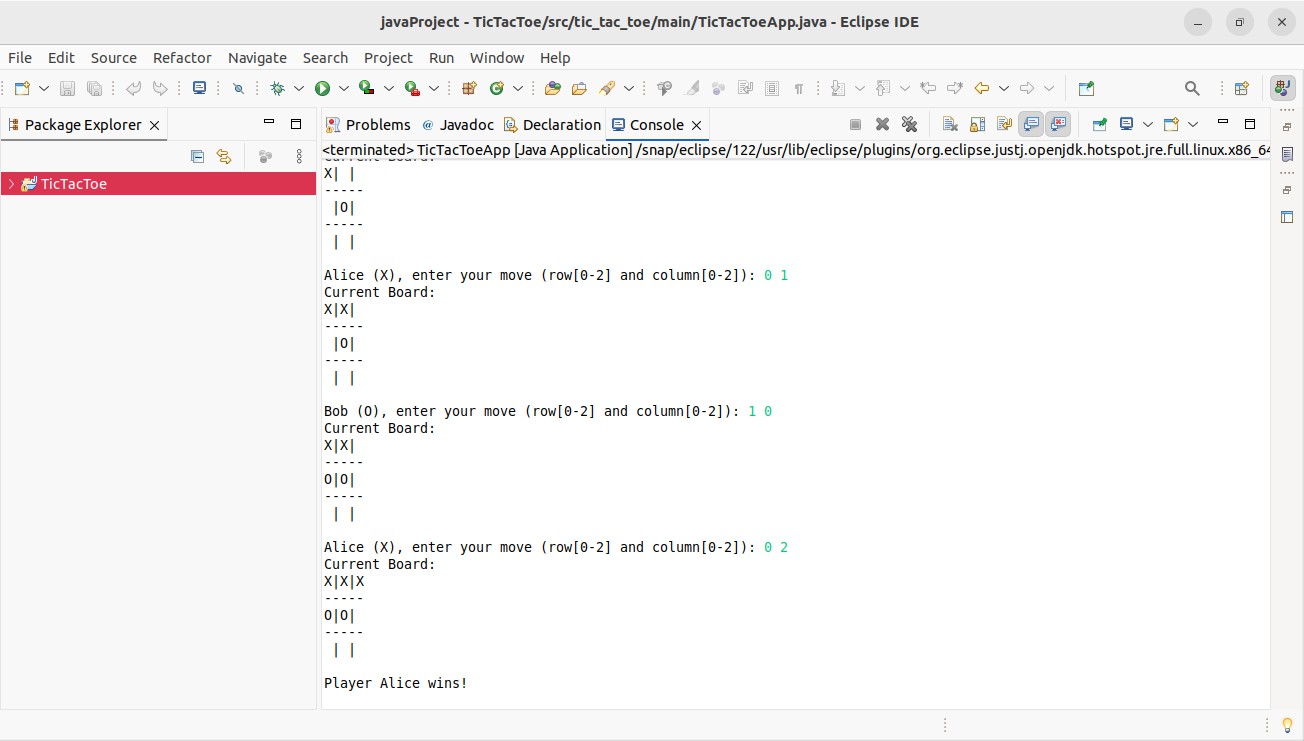
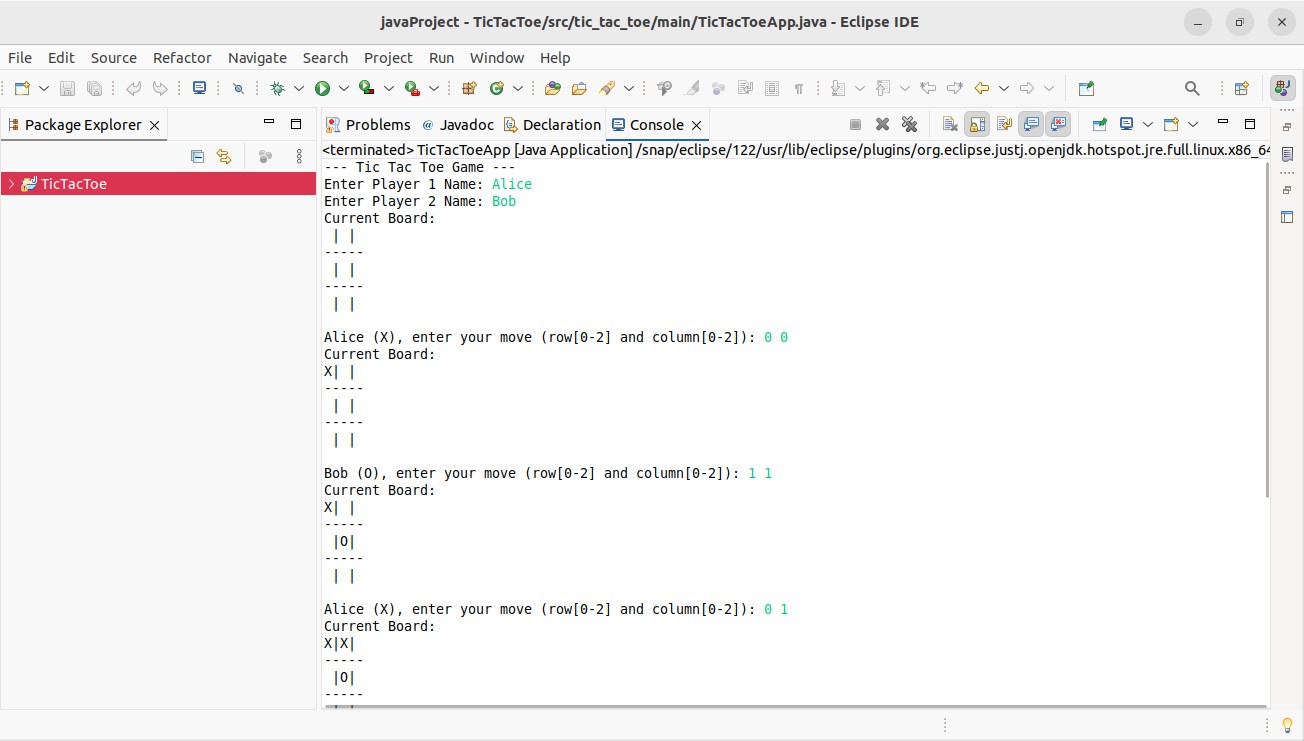
scanner.close();

System.out.println("Thank you for playing!");

}

}

* 1. **OUTPUT**



# ExceptionHandling

* + - A custom exception class: InvalidMoveException to handle illegal or out-of-bound moves.
    - A custom exception class: GameAlreadyOverException to prevent moves after the game ends.
    - Input validation to ensure only available cells are selected and coordinates are within range
    - Uses try-catch blocks to catch and display user-friendly error messages during gameplay.
    - Ensures smooth gameplay by preventing crashes due to invalid input or actions

# Conclusion

The Tic Tac Toe Game project offered hands-on experience with Java fundamentals such as classes, objects, exception handling, and modular programming. It reinforced core object-oriented principles like encapsulation and separation of concerns. The clean console- based design ensures clarity and usability, while the project structure supports easy maintenance and future upgrades such as score tracking or AI-based opponents

.

**Java Project 2: Number Guessing Game**

# Abstract

The Number Guessing Game is a console-based Java application where the user attempts to guess a randomly selected number within a predefined range. The game provides feedback after each guess, such as “Too high” or “Too low,” helping guide the player toward the correct number. The player is limited to a maximum of 7 attempts per round to promote strategic guessing. The application emphasizes clean input handling, exception management, and object-oriented design. It also supports replayability and can be extended with features like difficulty levels or score tracking.

# Introduction

This project introduces a simple yet interactive way to understand core Java programming concepts through the implementation of a number guessing game. It utilizes Java classes, objects, and exception handling to manage game flow and user inputs efficiently. The application provides a practical demonstration of loops, conditionals, random number generation, and modular design. It is intended for beginners to reinforce their understanding of console I/O and logic structures. The game is designed to be user-friendly, engaging, and extendable for future enhancements like AI hints or GUI interfaces.

# Objective of the Project

To develop an interactive Java console application that:

* + - Randomly generates a number and challenges the user to guess it.
    - Provides hints like "Too high" or "Too low" after each attempt.
    - Limits the number of attempts to promote accuracy and logic
    - Implements custom exception handling for invalid inputs.
    - Encourages replayability through a simple and engaging interface.

# Functional Requirements

## Input Handling:

* + Accepts user inputs for guesses.
  + Validates inputs to ensure it's a number within the specified range (1-100).
  + Uses custom exceptions for invalid entries.

## Game Logic:

* + Generates a random number at the start of each game.
  + Compares user guesses and provides hints.
  + Tracks remaining attempts (max 7 per round).

## Replay Functionality:

* + Asks the user if they want to play again after each game.
  + Resets the game if the user chooses to continue.

## Difficulty Levels (Optional) :

* + Easy: 1–50, Medium: 1–100, Hard: 1–200 (can be added for flexibility).

## Interface :

* + Menu-driven
  + Console-based
  + User-friendly Interactions

# Technologies Used

* + - Programming Language: Java
    - OOP Concepts: Encapsulation, Abstraction
    - Utility Classes: Random, Scanner
    - Collections: N/A
    - Exception Handling: Custom Exception (InvalidInputException), try-catch blocks.

# Project Structure

## Custom Exception: InvalidInputException

**Purpose**: Handles user input errors such as non-numeric or out-of-range values.

* + Extends Java’s Exception class
  + Displays custom error messsages

## Class: GuessGameService

**Purpose**: Contains core game logic such as generating numbers, processing guesses, and tracking attempts.

* + Fields: targetNumber, maxAttempts, currentAttempt
  + Methods:
    - startGame() : Initializes and runs the game
    - validateInput() : Validates and parses user input
    - checkGuess() : Compares user guess with the target number

## Main Application: NumberGuessGameApp

**Purpose**: Entry point of the application. Handles user interface and game flow.

* + Fields: Scanner
  + Main Method:
    - Calls the GuessGameService
    - Manages user input and game restart prompts.

# Implementation Details

**InvalidInputException.java**

package number\_guess\_game.exceptions;

public class InvalidInputException extends Exception { public InvalidInputException(String message) {

super(message);

}

}

**GuessGameService.java**

package number\_guess\_game.services; import java.util.Random;

import number\_guess\_game.exceptions.InvalidInputException; public class GuessGameService {

private int targetNumber; private int maxAttempts; private int attemptsMade; private int maxRange;

public GuessGameService(String difficulty) { this.maxAttempts = 7;

this.attemptsMade = 0;

this.maxRange = switch (difficulty.toLowerCase()) { case "easy" -> 50;

case "hard" -> 200; default -> 100;

};

generateNumber();

}

private void generateNumber() { Random rand = new Random();

this.targetNumber = rand.nextInt(maxRange) + 1;

}

public String processGuess(String input) throws InvalidInputException { int guess;

try {

guess = Integer.parseInt(input);

} catch (NumberFormatException e) {

throw new InvalidInputException("Input must be an integer.");

}

if (guess < 1 || guess > maxRange) {

throw new InvalidInputException("Guess must be between 1 and " + maxRange);

}

attemptsMade++;

if (guess == targetNumber) {

return "Correct! You guessed it in " + attemptsMade + " attempts.";

} else if (guess < targetNumber) { return "Too low!";

} else {

return "Too high!";

}

}

public boolean isGameOver() {

return attemptsMade >= maxAttempts;

}

public int getRemainingAttempts() { return maxAttempts - attemptsMade;

}

public int getTargetNumber() { return targetNumber;

}

public int getMaxRange() { return maxRange;

}

public int getAttemptsMade() { return attemptsMade;

}

public void reset(String difficulty) { this.attemptsMade = 0;

this.maxRange = switch (difficulty.toLowerCase()) { case "easy" -> 50;

case "hard" -> 200; default -> 100;

};

generateNumber();

}

}

**NumberGuessGameApp.java** package number\_guess\_game.main; import java.util.Scanner;

import number\_guess\_game.services.GuessGameService; import number\_guess\_game.exceptions.InvalidInputException; public class NumberGuessGameApp {

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

int bestScore = Integer.MAX\_VALUE; System.out.println("--- Number Guessing Game ---"); boolean playAgain = true;

while (playAgain) {

System.out.print("Select difficulty (easy / medium / hard): "); difficulty = scanner.nextLine().trim();

GuessGameService game = new GuessGameService(difficulty); System.out.println("Guess a number between 1 and " + game.getMaxRange()); while (!game.isGameOver()) {

System.out.print("Attempt " + (game.getAttemptsMade() + 1) + ": Enter your guess: "); String input = scanner.nextLine();

try {

String result = game.processGuess(input); System.out.println(result);

if (result.startsWith("Correct!")) {

if (game.getAttemptsMade() < bestScore) { bestScore = game.getAttemptsMade();

System.out.println(" ˛.‘~.ª~·˙3 New Best Score: " + bestScore + " attempts!");

}

break;

} else {

System.out.println("Remaining Attempts: " + game.getRemainingAttempts());

}

}

catch (InvalidInputException e) { System.out.println("Error: " + e.getMessage());

}

}

if (game.isGameOver() && game.getTargetNumber() != Integer.parseInt(scanner.nextLine())) { System.out.println("Game Over! The number was: " + game.getTargetNumber());

}

System.out.print("Do you want to play again? (yes/no): "); playAgain = scanner.nextLine().equalsIgnoreCase("yes");

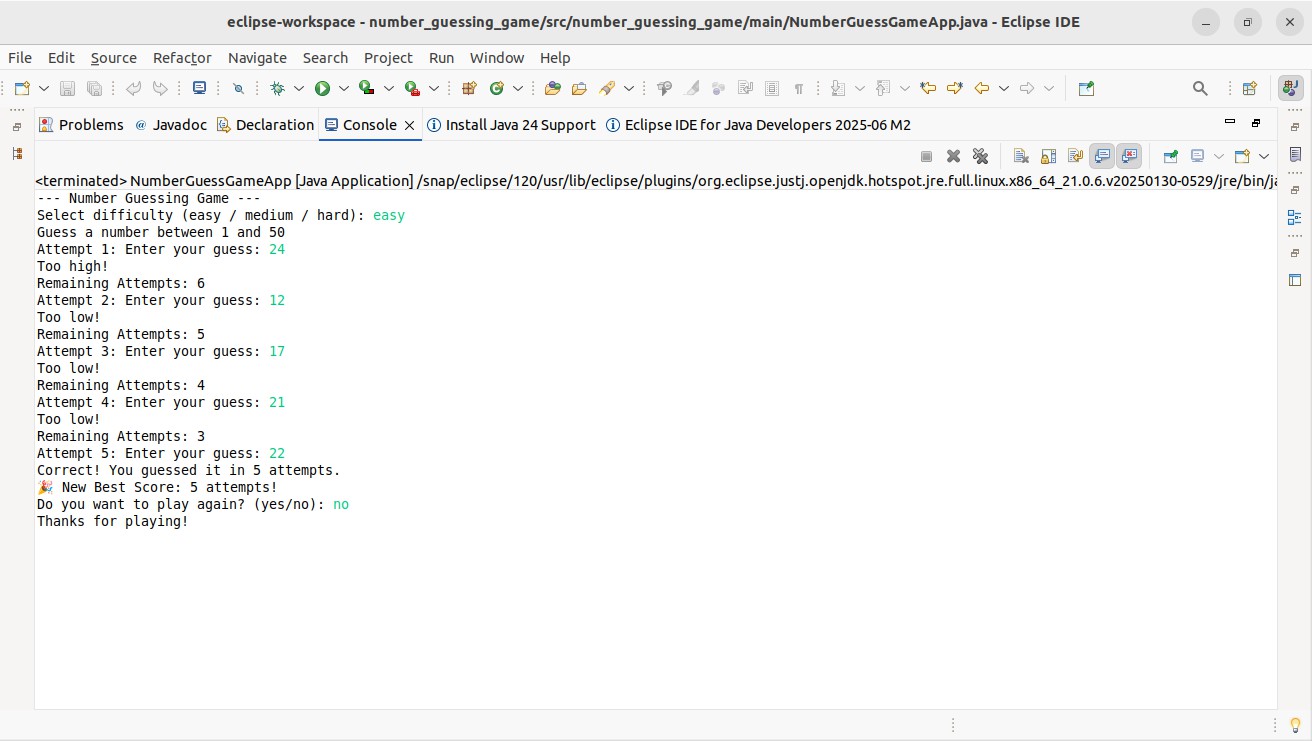
}

System.out.println("Thanks for playing!"); scanner.close();

}

}

* 1. **OUTPUT:**



# ExceptionHandling

* + - A custom exception class: InvalidInputException.
    - Input validation (e.g., checking for non-numeric or out-of-range input.
    - Catching NumberFormatException for non-integer user entries.
    - Prevents program crashes improves user experience.

# Conclusion

The Number Guessing Game project reinforced foundational Java programming skills, especially in control structures, input validation, and exception handling. Its simple, modular design makes it easy to extend with features like score tracking or difficulty modes. The interactive console interface offers an engaging user experience and demonstrates effective use of object-oriented principles.

# Outcomes

* + Demonstrates core OOP concepts like enacpsulation and modular class design.
  + Implements custom exception handling for robust input validation.
  + Offers an interactive console-based user experience with clear feedback.
  + Uses random number generation and conditional logic effectively.
  + Facilitates game replayability and potential for feature extension(e.g., difficulty levels, scoring).

# References

* <https://www.w3schools.com/java/>
* <https://www.geeksforgeeks.org/java/>
* https:/[/www.tpointtech.com/java-tutorial](http://www.tpointtech.com/java-tutorial)