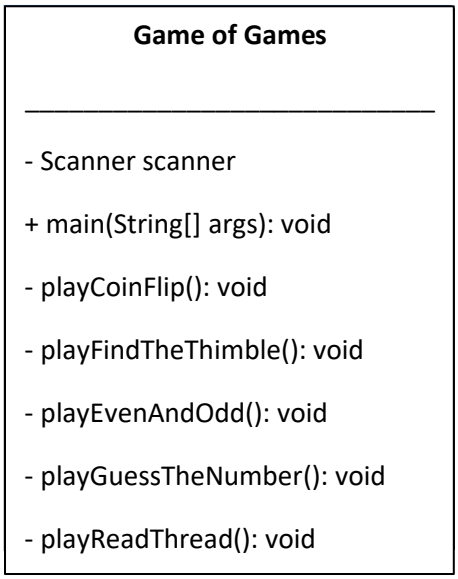
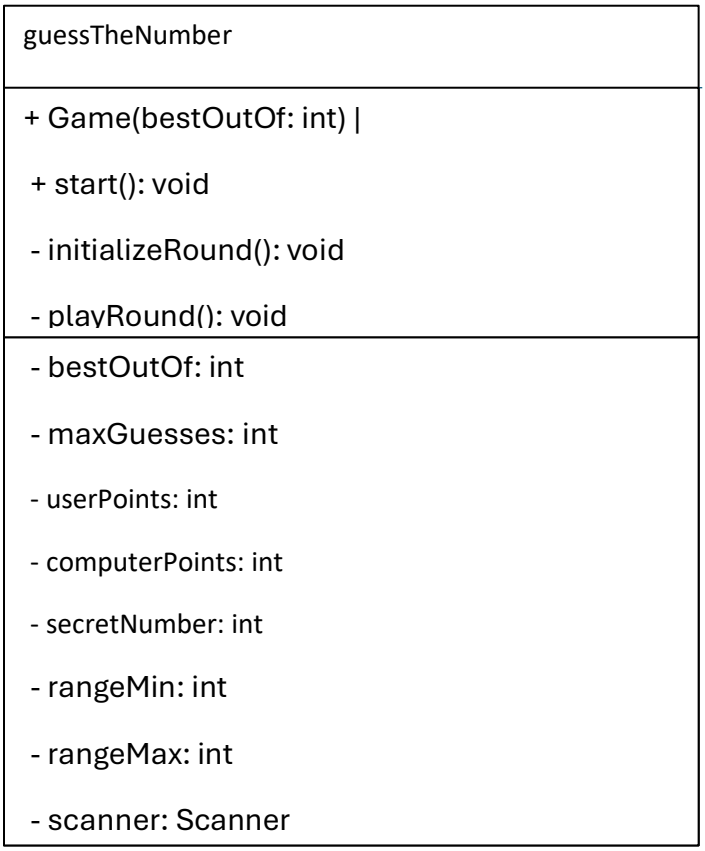
Table of Contents

Documentation

UML Diagram





EvenOddGame
- bestOf: int - winningPoints: int - userPoints: int - computerPoints: int
+ main(args: String[]): void - getBestOf(scanner: Scanner): int - getUserChoice(scanner: Scanner): String - getUserNumber(scanner: Scanner): int - getComputerNumber(random: Random): int - determineEvenOrOdd(sum: int): String - displayScore(userPoints: int, computerPoints: int): void - declareWinner(userPoints: int, winningPoints: int): void

FindTheThimble
- bestOutOf: int - userScore: int - computerScore: int
+ main(args: String[]): void - getValidBestOutOf(): int - playRound(): void - processGuess(guess: String, thimbleHand: String): void - resetRoundFlags(): void - displayFinalResult(int userScore, int computerScore): void

FindTheRedThread
- totalThreads: int - spoolsPerTurn: int - isPlayerTurn: boolean
+ main(args: String[]): void - getTotalThreads(scanner: Scanner): int - generateThreads(totalThreads: int, random: Random): Map<Integer, String> - getSpoolsPerTurn(scanner: Scanner, totalThreads: int): int - decideFirstTurn(scanner: Scanner): boolean

- playerTurn(scanner: Scanner, threads: Map<Integer, String>, spoolsPerTurn: int): boolean - computerTurn(threads: Map<Integer, String>, random: Random, spoolsPerTurn: int): boolean }

GameofGames
+ main(args: String[]): void - displayMainMenu(): void - findTheThimble(): void - coinFlip(): void - evenOrOdd(): void - findTheRedThread(): void - guessTheNumber(): void - displayFinalScoreboard(): void
+ main(args: String[]): void + testSunnyDayPaths(): void + testRainyDayPaths(): void

CoinFlip Class Method Glossary

Constructor

CoinFlip(int roundsToWin)

- **Purpose:** Initializes a new CoinFlip game instance
- **Access Modifier:** public
- **Parameters:**
    - roundsToWin (int): The number of rounds needed to win the game
- **Initialization:**
    - Sets userScore to 0
    - Sets computerScore to 0
    - Sets the roundsToWin parameter
    - Initializes Random object for coin flipping

Game Logic Methods

flipCoin()

- **Purpose:** Simulates flipping a coin
- **Access Modifier:** public
- **Parameters:** None
- **Returns:** String ("Heads" or "Tails")
- **Behavior:**
    - Uses Random.nextBoolean() to generate random result
    - Converts Boolean to "Heads" or "Tails"

playRound(String userGuess)

- **Purpose:** Processes a single round of the game
- **Access Modifier:** public
- **Parameters:**
    - userGuess (String): Player's prediction ("Heads" or "Tails")
- **Returns:** Boolean (true if guess was correct)
- **Behavior:**
    - Flips the coin
    - Compares result with user's guess
    - Updates appropriate score (user or computer)
    - Displays round result

Game State Methods

isGameOver()

- **Purpose:** Checks if either player has won the game
- **Access Modifier:** public
- **Parameters:** None
- **Returns:** boolean
    - true: if either player has reached roundsToWin
    - false: if game is still ongoing
- **Behavior:** Compares current scores against roundsToWin

displayScores()

- **Purpose:** Shows current game standings

- **Access Modifier:** public
- **Parameters:** None
- **Returns:** void
- **Behavior:**
    - Prints user's current score
    - Prints computer's current score

getWinner()

- **Purpose:** Determines the winner of the game
- **Access Modifier:** public
- **Parameters:** None
- **Returns:** String ("User" or "Computer")
- **Behavior:**
    - Compares userScore and computerScore
    - Returns winner based on higher score

Instance Variables

- **userScore:** int
    - Tracks player's wins
    - Initialized to 0
    - Incremented when player guesses correctly
- **computerScore:** int
    - Tracks computer's wins
    - Initialized to 0
    - Incremented when player guesses incorrectly
- **roundsToWin:** int
    - Stores number of rounds needed to win
    - Set via constructor
    - Used to determine game end
- **random:** Random
    - Random number generator for coin flips
    - Initialized in constructor
    - Used by flipCoin() method

GuessTheNumber class method glossary

Constructor

Game(int bestOutOf)

- **Purpose:** Initializes a new GuessTheNumber game instance
- **Access Modifier:** public
- **Parameters:**
    - bestOutOf (int): The number of rounds needed to win the game
- **Initialization:**
    - Sets userPoints to 0
    - Sets computerPoints to 0
    - Sets maxGuesses to 5
    - Creates a new Scanner for user input

Game Logic Methods

initializeRound()

- **Purpose:** Prepares a new round by generating a random number range
- **Access Modifier:** private
- **Parameters:** None
- **Behavior:**
    - Generates a random minimum range value between 1 and 50
    - Creates a maximum range value ensuring at least 10 possible numbers
    - Randomly selects a secret number within the generated range
    - Prints the current round's number range to the user

playRound()

- **Purpose:** Manages a single round of number guessing
- **Access Modifier:** private
- **Parameters:** None
- **Behavior:**
    - Provides user with a fixed number of guesses (5)
    - Validates user's guess is within the current round's range
    - Provides feedback if the guess is too high or too low
    - Increments user points for a correct guess
    - Increments computer points if user fails to guess within max attempts

Game State Methods

start()

- **Purpose:** Controls the overall game flow
- **Access Modifier:** public
- **Parameters:** None
- **Behavior:**
    - Runs game rounds until a player reaches bestOutOf points
    - Calls initializeRound() for each new round
    - Calls playRound() to process individual rounds
    - Ends game by calling displayFinalScore()

displayFinalScore()

- **Purpose:** Shows the final game results
- **Access Modifier:** private
- **Parameters:** None
- **Behavior:**
    - Prints total points for user and computer
    - Determines and announces the winner
    - Displays a congratulatory or encouraging message

Instance Variables

- **bestOutOf:** int
    - Stores the number of rounds needed to win
    - Set via constructor
    - Used to determine game end
- **maxGuesses:** int
    - Fixed number of guesses per round
    - Initialized to 5
- **userPoints:** int
    - Tracks player's wins
    - Initialized to 0
    - Incremented when player guesses correctly
- **computerPoints:** int
    - Tracks computer's wins
    - Initialized to 0

    - Incremented when player fails to guess
- **secretNumber:** int
    - Stores the randomly generated number to guess
    - Generated in initializeRound()
- **rangeMin:** int
    - Minimum value of the guessing range
    - Randomly generated for each round
- **rangeMax:** int
    - Maximum value of the guessing range
    - Randomly generated for each round
- **scanner:** Scanner
    - Handles user input
    - Initialized in constructor

Even Odd Class Method Glossary

Game Logic Methods

main(String[] args): void

- **Purpose**: Entry point of the game. Sets up the game, handles the main game loop, and declares the winner.
- **Access Modifier**: public
- **Parameters**:
    - args (String[]): Command-line arguments (not used in the implementation).
- **Behavior**:
    - Initializes Scanner and Random.
    - Calls game setup methods: getBestOf, getUserChoice, getUserNumber, etc.
    - Alternates turns between the player and computer.
    - Tracks and displays scores after each round.
    - Ends the game when either player reaches winningPoints.

Helper Methods

getBestOf(Scanner scanner): int

- **Purpose**: Prompts the user to enter the "best of" value for the game.
- **Access Modifier**: private

- **Parameters**:
    - scanner (Scanner): Used to capture user input.
- **Returns**: The odd "best of" number input by the user.
- **Behavior**:
    - Ensures the input is a positive, odd number.
    - Loops until a valid value is provided.

getUserChoice(Scanner scanner): String

- **Purpose**: Prompts the user to choose "odd" or "even."
- **Access Modifier**: private
- **Parameters**:
    - scanner (Scanner): Used to capture user input.
- **Returns**: The user's choice ("odd" or "even").
- **Behavior**:
    - Validates input to ensure it's either "odd" or "even."
    - Re-prompts the user for invalid inputs.

getUserNumber(Scanner scanner): int

- **Purpose**: Prompts the user to enter a positive number for the round.
- **Access Modifier**: private
- **Parameters**:
    - scanner (Scanner): Used to capture user input.
- **Returns**: The positive number input by the user.
- **Behavior**:
    - Validates that the input is greater than zero.
    - Re-prompts the user for invalid inputs.

getComputerNumber(Random random): int

- **Purpose**: Generates a random number for the computer's turn.
- **Access Modifier**: private
- **Parameters**:
    - random (Random): Random number generator instance.
- **Returns**: A random integer between 1 and 10.
- **Behavior**:

- Uses Random.nextInt to generate the number.

determineEvenOrOdd(int sum): String

- **Purpose**: Determines whether a given number is "even" or "odd."
- **Access Modifier**: private
- **Parameters**:
  - sum (int): The number to check.
- **Returns**: "even" if the number is even, "odd" otherwise.
- **Behavior**:
  - Checks divisibility by 2.

displayScore(int userPoints, int computerPoints): void

- **Purpose**: Displays the current scores for the user and computer.
- **Access Modifier**: private
- **Parameters**:
  - userPoints (int): The user's current score.
  - computerPoints (int): The computer's current score.
- **Returns**: void
- **Behavior**:
  - Prints the scores in the format: Score: You X - Computer Y.

declareWinner(int userPoints, int winningPoints): void

- **Purpose**: Declares the winner of the game.
- **Access Modifier**: private
- **Parameters**:
  - userPoints (int): The user's score.
  - winningPoints (int): The threshold points required to win.
- **Returns**: void
- **Behavior**:
  - Prints a congratulatory message if the user wins.
  - Prints a loss message if the computer wins.

Instance Variables:

- bestOf: int
    - **Purpose**: Stores the "best of" value input by the user.
    - **Initialization**: Set via the getBestOf method.
    - **Behavior**: Determines the number of rounds required to decide a winner.
- winningPoints: int
    - **Purpose**: Stores the number of points needed to win the game.
    - **Initialization**: Calculated as (bestOf / 2) + 1 at the start of the game.
    - **Behavior**: Tracks the threshold for ending the game.
- userPoints: int
    - **Purpose**: Tracks the user's score.
    - **Initialization**: Initialized to 0.
    - **Behavior**: Incremented when the user wins a round.
- computerPoints: int
    - **Purpose**: Tracks the computer's score.
    - **Initialization**: Initialized to 0.
    - **Behavior**: Incremented when the computer wins a round.

Find the Thimble methods glossary

main(String[] args): void

- **Purpose:** The entry point of the program. Initializes the game, handles the game loop, and displays the final result.
- **Parameters:** args (String[]): Command-line arguments (not used in this program).
- **Return:** None.
- **Behavior:**
- Welcomes the user to the game.
- Gets the desired number of rounds (bestOutOf).
- Initializes userScore and computerScore to 0.
- Starts the game loop:
    - Resets round flags.
    - Plays a round.
    - Updates scores based on the round result.
- Continues the loop until a player reaches the required number of wins.
- Displays the final scores and winner.

Game Logic Methods

- **getValidBestOutOf()**:
    - **Purpose:** Prompts the user for the number of rounds needed to win and validates the input.
    - **Parameters:** None.
    - **Return:** An integer representing the valid bestOutOf value.
    - **Behavior:**
        - Prompts the user for input.
        - Validates the input to ensure it's an odd positive integer.
        - Returns the valid input or prompts the user again if invalid.
- **playRound()**:
    - **Purpose:** Handles a single round of the game.
    - **Parameters:** None.
    - **Return:** None.
    - **Behavior:**
        - Generates a random hand for the thimble.
        - Starts a timer for user input.
        - Gets user's guess.
        - Processes the guess and updates scores.
        - Handles timeouts and invalid input.
- **processGuess(guess: String, thimbleHand: String)**:
    - **Purpose:** Compares the user's guess to the actual thimble hand and updates the round result.
    - **Parameters:**
        - guess: The user's guess ("left" or "right").
        - thimbleHand: The correct hand with the thimble ("left" or "right").
    - **Return:** None.
    - **Behavior:**
        - Compares the guess and thimbleHand.
        - Updates the roundResult flag accordingly.

- **resetRoundFlags()**:
    - **Purpose:** Resets the round-specific flags for the next round.
    - **Parameters:** None.
    - **Return:** None.
    - **Behavior:**
        - Resets timedOut, inputComplete, and roundResult flags.

- **displayFinalResult(userScore: int, computerScore: int)**:
    - **Purpose:** Displays the final scores and declares the winner.
    - **Parameters:**
        - userScore: The final score of the user.
        - computerScore: The final score of the computer.
    - **Return:** None.
    - **Behavior:**
        - Prints the final scores.
        - Determines the winner based on the scores.
        - Prints a congratulatory or encouraging message.

Find the red thread methods glossary

main(String[] args): void

- **Purpose:** The entry point of the program. Initializes the game, handles the game loop, and displays the final result.
- **Parameters:** args (String[]): Command-line arguments (not used in this program).
- **Return:** None.
- **Behavior:**
    - Initializes the game by getting the total number of threads and the number of spools per turn from the user.
    - Decides who goes first (player or computer).
    - Starts the game loop:
        - Alternates between player and computer turns.
        - Checks for game-ending conditions (finding the red thread or running out of threads).
    - Displays the final result (win, loss, or draw).

Game Logic Methods

- **getTotalThreads(scanner: Scanner): int:**
    - **Purpose:** Prompts the user to enter the total number of threads.
    - **Parameters:** scanner (Scanner): A Scanner object for user input.
    - **Return:** An integer representing the total number of threads.
    - **Behavior:**

        - Repeatedly prompts the user until a valid number between 2 and 100 is entered.
- **generateThreads(totalThreads: int, random: Random): Map<Integer, String>:**
    - **Purpose:** Generates a map of threads with random colors, including one red thread.
    - **Parameters:**
        - totalThreads: The total number of threads.
        - random: A Random object for generating random numbers.
    - **Return:** A Map<Integer, String> representing the threads and their colors.
    - **Behavior:**
        - Creates a new LinkedHashMap to store threads and their colors.
        - Randomly selects a key for the red thread.
        - Iterates through the total number of threads:
            - If the current key is the red thread's key, assigns the "red" color.
            - Otherwise, assigns a random color from a predefined list.
- **getSpoolsPerTurn(scanner: Scanner, totalThreads: int): int:**
    - **Purpose:** Prompts the user to enter the number of spools to pick per turn.
    - **Parameters:**
        - scanner: A Scanner object for user input.
        - totalThreads: The total number of threads.
    - **Return:** An integer representing the number of spools per turn.
    - **Behavior:**
        - Repeatedly prompts the user until a valid number between 1 and half the total threads is entered.
- **decideFirstTurn(scanner: Scanner): boolean:**
    - **Purpose:** Determines who goes first (player or computer) based on user input.
    - **Parameters:** scanner: A Scanner object for user input.
    - **Return:** A boolean indicating whether the player goes first.
    - **Behavior:**
        - Prompts the user to choose whether to go first.
        - Returns true if the user chooses to go first, false otherwise.
- **playerTurn(scanner: Scanner, threads: Map<Integer, String>, spoolsPerTurn: int): boolean:**
    - **Purpose:** Handles the player's turn.
    - **Parameters:**
        - scanner: A Scanner object for user input.

        - threads: A Map representing the remaining threads.
        - spoolsPerTurn: The number of threads the player can pick.
    - **Return:** A boolean indicating whether the game is over (true if the red thread is found or a draw occurs).
    - **Behavior:**
        - Prompts the player to choose keys for the threads to pick.
        - Validates the user's choices.
        - Removes the chosen threads from the map.
        - Checks if the red thread was picked or if there's a draw.
- **computerTurn(threads: Map<Integer, String>, random: Random, spoolsPerTurn: int): boolean:**
    - **Purpose:** Handles the computer's turn.
    - **Parameters:**
        - threads: A Map representing the remaining threads.
        - random: A Random object for generating random choices.
        - spoolsPerTurn: The number of threads the computer can pick.
    - **Return:** A boolean indicating whether the game is over (true if the computer finds the red thread or a draw occurs).
    - **Behavior:**
        - Randomly selects keys from the available threads.
        - Removes the chosen threads from the map.
        - Checks if the computer picked the red thread or if there's a draw.

GameOfGames methods glossary

Game logic methods

main (String[] args)

- **Purpose**: Entry point of the program. Continuously displays the main menu and allows the user to select different games or exit.
- **Parameters**:
    - String[] args: Command-line arguments (unused in this implementation).
- **Returns**: void
- **Key Functionality**:
    - Controls the main game loop.
    - Calls appropriate methods based on user input.

displayMainMenu()

- **Purpose**: Displays the main menu options for the user.
- **Parameters**: None
- **Returns**: void
- **Key Functionality**:
    - Prints the menu options to the console.

findTheThimble()

- **Purpose**: Simulates the "Find the Thimble" game.
- **Parameters**: None
- **Returns**: void
- **Key Functionality**:
    - Placeholder for game logic.

coinFlip()

- **Purpose**: Simulates a coin flip game.
- **Parameters**: None
- **Returns**: void
- **Key Functionality**:
    - Placeholder for game logic.

evenOrOdd()

- **Purpose**: Simulates a game where the user guesses if a number is even or odd.
- **Parameters**: None
- **Returns**: void
- **Key Functionality**:
    - Placeholder for game logic.

findTheRedThread()

- **Purpose**: Simulates the "Find the Red Thread" game.
- **Parameters**: None
- **Returns**: void
- **Key Functionality**:

- Placeholder for game logic.

guessTheNumber()

- **Purpose**: Simulates a number-guessing game.
- **Parameters**: None
- **Returns**: void
- **Key Functionality**:
  - Placeholder for game logic.

displayFinalScoreboard()

- **Purpose**: Displays the final scoreboard before exiting the game.
- **Parameters**: None
- **Returns**: void
- **Key Functionality**:
  - Placeholder for displaying final scores or statistics.

main(String[] args)

- **Purpose**: Entry point for running tests. Executes both sunny day and rainy-day path tests.
- **Parameters**:
  - String [] args: Command-line arguments (unused in this implementation).
- **Returns**: void
- **Key Functionality**:
  - Calls test methods to validate functionality.

testSunnyDayPaths()

- **Purpose**: Tests scenarios where everything works as expected (sunny day paths).
- **Parameters**: None
- **Returns**: void
- **Key Functionality**:
  - Validates each game method and menu functionality in normal conditions.

testRainyDayPaths()

- **Purpose**: Tests scenarios where errors or unexpected issues occur (rainy day paths).

- **Parameters**: None
- **Returns**: void
- **Key Functionality**:
    - Simulates failures for specific methods and ensures recovery mechanisms.

Coin Flip – Data Configuration Table

Value & Variable	Input Format	Output Type/Format	Internal Representation
User Score (userScore)	N/A	Integer value, displayed as "User Score: X".	Stored as an integer, incremented when the user guesses correctly.
Computer Score (computerScore)	N/A	Integer value, displayed as "Computer Score: X".	Stored as an integer, incremented when the user guesses incorrectly.
Rounds to Win (roundsToWin)	Integer provided as a parameter to the constructor.	Integer value indicating the rounds required to win.	Stored as an integer variable, determines the game's winning condition.
Coin Result (result)	N/A (Randomly generated internally using flipCoin()).	String: "Heads" or "Tails".	Randomly selected using Random.nextBoolean().
User Guess (userGuess)	User inputs a string: "Heads" or "Tails" (case-insensitive).	String: "Heads" or "Tails".	Provided as a parameter to playRound().
Game Status (isGameOver())	N/A	Boolean: true if game is over, false otherwise.	Evaluated based on scores (userScore and computerScore).
Winner (getWinner())	N/A	String: "User" or "Computer".	Computed by comparing userScore and computerScore.

Guess The Number – Data Configuration Table

Value & Variable	Input Format	Output Type/Format	Internal Representation
Best Out Of (bestOutOf)	Positive odd integer input by the user.	Integer value defines the total rounds.	Stored as an integer in the Game constructor.
Maximum Guesses (maxGuesses)	Fixed value of 5 (predefined in code).	Integer value representing guesses per round.	Stored as an integer constant in the Game class.
User Points (userPoints)	N/A	Integer value displayed as "User Points: X".	Incremented when the user guesses the number correctly.
Computer Points (computerPoints)	N/A	Integer value displayed as "Computer Points: X".	Incremented when the user runs out of guesses.
Range Minimum (rangeMin)	Random integer generated between 1 and 50.	Integer value representing the lower bound of the range.	Generated in the initializeRound() method.
Range Maximum (rangeMax)	Random integer at least 10 greater than rangeMin.	Integer value representing the upper bound of the range.	Generated in the initializeRound() method.
Secret Number (secretNumber)	Random integer within rangeMin and rangeMax.	Integer value hidden from the user.	Generated in the initializeRound() method.
User Guess (userGuess)	Positive integer input by the user within the range.	Integer value representing the user's guess.	Collected as an integer during each turn.
Game Status	N/A	Boolean: Ends when either user or computer reaches the bestOutOf points.	Managed in the start() loop.
Winner	N/A	String: "Congratulations! You won" or "Computer won".	Determined by comparing userPoints and computerPoints.

Even Odd Game – Data Configuration Table

Value & Variable	Input Format	Output Type/Format	Internal Representation
Best of Rounds (bestOf)	Positive odd integer input by the user.	Integer value defines the total rounds.	Stored as an integer, validated in getBestOf().
User Choice (userChoice)	String input: "odd" or "even" (case-insensitive).	String: "odd" or "even".	Provided as a lowercase string from getUserChoice().
User Number (userNumber)	Positive integer input by the user.	Integer value between 1 and ∞.	Stored as an integer from getUserNumber().
Computer Number (computerNumber)	Random integer between 1 and 10.	Integer value between 1 and 10.	Generated using Random.nextInt(10) + 1.
Sum of Numbers (sum)	Computed as the sum of userNumber and computerNumber.	Integer value represents the sum.	Calculated and stored temporarily.
Sum Result (result)	Derived as "even" or "odd" based on sum.	String: "even" or "odd".	Determined by checking sum % 2 in determineEvenOrOdd().
User Points (userPoints)	N/A	Integer value displayed as "You X - Computer Y".	Incremented when the user wins a round.
Computer Points (computerPoints)	N/A	Integer value displayed as "You X - Computer Y".	Incremented when the computer wins a round.
Winning Points (winningPoints)	Derived as (bestOf / 2) + 1.	Integer value representing the score needed to win.	Calculated at the start of the game.
Game Status (isGameOver)	N/A	Boolean: Game ends when either	Managed the game loop.

		score equals winningPoints.	
Winner (declareWinner)	N/A	String: "Congratulations! You won" or "Computer wins".	Determined based on points comparison.

Find The Read Thread – Data Configuration Table

Value & Variable	Input Format	Output Type/Format	Internal Representation
Total Threads (totalThreads)	Integer input by the user, restricted between 2 and 100.	Integer value, representing the total number of threads.	Stored as an integer variable in the main() method.
Spools Per Turn (spoolsPerTurn)	Integer input by the user, restricted between 1 and half of totalThreads.	Integer value, representing the threads picked per turn.	Stored as an integer variable.
Threads (threads)	N/A (Generated internally: each thread is assigned a color, including one red thread).	Displayed as a map of thread numbers and their colors	Stored as a Map<Integer, String> (key: thread number, value: color).
Red Thread (redThreadKey)	Randomly generated integer between 1 and totalThreads.	N/A	Stored as an integer value in the generateThreads() method.
Player Choice (chosenKey)	Integer input by the user, restricted to valid keys in the threads map.	String, displaying the chosen thread's color (e.g., "red").	Stored in a Set<Integer> during the player's turn.
Computer Choice (chosenKey)	Randomly selected integer key from the threads map.	String, displaying the chosen thread's color (e.g., "red").	Managed as a random choice from a List<Integer> of available keys.
Thread Colors (colors)	N/A (Randomly assigned from a predefined set).	Strings ("blue", "green", "yellow", "red").	Stored as a String[] and selected

			randomly in generateThreads().

Find The Thimble – Data Configuration Table

Value & Variable	Input Format	Output Type/Format	Internal Representation
Best Out Of (bestOutOf)	Positive odd integer entered by the user.	Integer defining the total rounds needed to win.	Stored as an integer in getValidBestOutOf().
Required Wins (requiredWins)	Computed as (bestOutOf / 2) + 1.	Integer defining the minimum wins required to end the game.	Computed once at the start of the game.
User Score (userScore)	N/A	Integer displayed as "You: X".	Incremented when the user guesses correctly.
Computer Score (computerScore)	N/A	Integer displayed as "Computer: Y".	Incremented when the user guesses incorrectly or times out.
Thimble Hand (thimbleHand)	Randomly assigned as "left" or "right".	String: "left" or "right".	Randomized using Random.nextBoolean().
User Guess (guess)	String input: "left" or "right" (case-insensitive).	String: "left" or "right".	Collected and validated during each round.
Timed Out (timedOut)	N/A	Boolean: true if the user fails to guess within 60 seconds.	Controlled by a TimerTask.
Input Complete (inputComplete)	N/A	Boolean: true if the user provides a valid guess.	Controlled by user input logic.
Game Status (gameOver)	N/A	Boolean: Ends the game when a player reaches the required wins.	Set to true when the game concludes.
Round Result (roundResult)	N/A	Boolean: true for a correct guess, false for incorrect, or null for timeout.	Managed at the end of each round.

Game of Games – Data Configuration Table

Value & Variable	Input Format	Output Type/Format	Internal Representation
Main Menu Choice (choice)	Integer input by the user.	Integer: Options from 1 to 6, representing game selections or exit.	Collected via Scanner.nextInt() and processed in a switch statement.
Exit Flag (exitGame)	Boolean, initialized as false.	Boolean: true to exit the game.	Set to true when the user selects option 6.
Game Names	Hardcoded Strings in the menu options.	Strings: Displayed as menu text (e.g., "Find the Thimble").	Stored in displayMainMenu() method.
Sub-Games	Calls to other methods representing games (e.g., findTheThimble).	Sub-game logic executed when respective menu options are selected.	Implemented as static methods in GameOfGames.
Final Scoreboard (displayFinalScoreboard)	N/A	String: Displays a summary of scores.	Placeholder method for future scoreboard logic.

Responsibility Document

A table outlining all responsibilities assigned to achieve the assignment goals is available. For detailed information, please refer to the Main Table and Gantt chart on Monday.com

Name	Accountability
Create a GitHub Repository	Joachim
Review assignment as a group during meeting	Myri, Joachim, Dago, Benedicte
Meeting Note update	Myri
Even Odd Implementation	Myri
Find the Thimble Implementation	Baile Benedicte
Find the Red Thread Implementation	Dagoberto Tellez, Baile Benedicte
Flip-coin game Implementation	Dagoberto Tellez
Guess the number Implementation	Joachim
Games of Games - Overall Implementation	Myri, Joachim
Readme on Github	Myri
UML for Coin-Flip	Myri
UML for Even-Odd	Dago
UML for Find the Thimble	Benedicte
UML for Read Thread	Dago, Benedicte
UML for Guess the Number	Joachim
CoinFlip Class Method Glossary	Myri
Even-Odd Class Method Glossary	Dago
Guess the Number Class Method Glossary	Joachim
Find the Red Thread Class Method Glossary	Dago, Benedicte
Find the Thimble Class Method Glossary	Benedicte
Game of Games Class Method Glossary	Myri, Joachim
Meeting Note #2	Dago
Coin Flip Test Case - Code Variation	Myri
Even-Odd Test Case - Code Variation	Dago
Guess the Number Test Case - Code Variation	Joachim

Find the Red Thread - Test Case - Code Variation	Dago, Benedicte
Find the Thimble - Test Case - Code Variation	Benedicte
Game of Games - Test Case - Code Variation	Myri, Joachim
Create a Responsibility Document	Myri
Create a Deployment Document	TBD
Create a Table of Content and Organize document	Joachim
Prepare and ZIP documents for Submission	Dago

Deployment Document

1. Accessing the Code

The code for the program is provided in zip files. Ensure you have received the following files:

- **PlayerMode.zip**
- **TestMode.zip**

Each zip file contains the required source code and associated resources for the respective mode.

2. Downloading the Code

To access the code:

1. Retrieve the zip files from the designated source (e.g., class resources folder or email attachment).
2. Save the zip files to a folder on your computer where you can easily locate them.

3. Setting Up the Environment

This program is written in Java. To run it, you need the following:

- **Java Development Kit (JDK):** Version 11 or higher. Download JDK if it is not installed.
- **Integrated Development Environment (IDE):** Recommended IDEs are IntelliJ IDEA, Eclipse, or NetBeans. Ensure your IDE is configured to work with the installed JDK.
- **Git:** (Optional) If using a version control system for future enhancements.

4. Configuring the Program

No special configuration is required. However, ensure:

1. You have extracted the zip files to separate folders (e.g., "PlayerMode" and "TestMode").
2. All .java files and associated resources are intact after extraction.

5. Running the Program

Option 1: Running in the Terminal

1. **Navigate to the Directory:** Open your terminal or command prompt and navigate to the folder containing the .java files (e.g., cd PlayerMode).
2. **Compile the Code:**

javac *.java

3. **Run the Program:**

java PlayGames

    a. For **Player Mode**, execute the program normally.
    b. For **Test Mode**, repeat these steps for the TestMode directory.

Option 2: Running in an IDE

1. **Open the IDE:** Launch your Java IDE.
2. **Import the Project:**
    a. Select "File" > "Open Project" (or equivalent in your IDE).
    b. Navigate to the extracted folder (e.g., "PlayerMode" or "TestMode") and select it.
3. **Build the Project:**
    a. Ensure there are no errors by building the project. Use the "Build" option in your IDE.
4. **Run the Program:**
    a. Locate the PlayGames class (the driver class).
    b. Right-click and select "Run" (or equivalent in your IDE).

6. Troubleshooting

- **Error: "Java not recognized":** Ensure JDK is installed and added to your system PATH.
- **Error: "Main class not found":** Verify that the PlayGames class is correctly marked as the main class.
- **Build Errors:** Double-check that all required files were extracted and imported into the IDE.
- **IDE Configuration Issues:** Consult your IDE’s documentation for resolving setup problems.

7. Additional Notes

- Be sure to use the provided zip files without modifying their contents unless instructed.
- Refer to the included documentation (UML diagrams, data tables, etc.) for further understanding of the program's design.

Project: Games of Games – Meeting Note

Meeting Number:	Meeting Date:	Start Time:	End Time:
1	07 December 2024	**6:15pm**	**6:45 pm**

Attendee Name (Indicate note taker with *)	Timeliness (on time, late, left early, missing)
Benedicte Baile	On time
Dago Tellez	On time
Joachim Chuah	On time
Myri Ayala*	On time

Purpose:

Schedule a meeting to review the assignment, divide tasks, decide how to approach it, agree on key points, and clarify any questions to ask others or the professor.

Agenda:

- Review the assignment description together.
- Decide who will implement each game.
- Assign someone to update the project management platform.
- Assign a primary contact for Slack questions.
- Share the GitHub repository and add all team members.
- Set an internal deadline for revisions.
- Pick a day for documentation updates.

Major Accomplishments:

- Benedicte:
    - Find the Thimble | Find the Red Thread
- Dago
    - Even-Odd | Find the Red Thread
- Joachim
    - Guess the number | Overall game of games
- Myri
    - Coin-Flip | Overall game of games

- Dago set up a new account and project on Monday.com to replace the current one, which is expiring. Dago and Joachim transferred and updated all information on the new project management platform.
- Dago and Benedicte were designated as the primary communicators on the Slack channel to address questions from members of other teams.
- Joachim created a GitHub repository and added all team members to it.
- The team agreed to push game implementations to individual branches instead of the main branch by Tuesday, the 10th, to avoid conflicts.
- Clarification was sought from the professor regarding the testing requirements for the project.

Next Steps:

Each member will individually implement their assigned games in Java. We'll meet on Wednesday at 4:30 pm at Zoom to address any team needs. Wednesday will also be our day for documentation progress.

Meeting Notes:

In the meeting, we reviewed the assignment, assigned games to each member, and set up a new Monday.com account for project management. Joachim created the GitHub repository, and we agreed to push code to individual branches. Dago and Benedicte were assigned as Slack communicators, and Wednesday was set as the day for documentation updates. We also reached out to the professor for clarification on testing requirements.

Documents

Testing_Document.docx [Tarek's group assignment]

Project: Games of Games – Meeting Note

Meeting Number:	Meeting Date:	Start Time:	End Time:
2	11 December 2024	5:30pm	6:15 pm

Attendee Name (Indicate note taker with *)	Timeliness (on time, late, left early, missing)
Benedicte Baile	On time
Dago Tellez*	On time
Joachim Chuah	On time
Myri Ayala	On time

Purpose:

Schedule a meeting to review the current progress, discuss what else needs to be done per person, decide how to approach it, agree on key points, and clarify any questions to ask others.

Agenda:

- Review the current code progress
- Decide and assign who will implement each documentation (Responsibility, Deployment, etc.)
- Understanding of different game implementation (player, test)
- Start adding all games together
- Set an internal deadline for final revisions.
- Pick a day for documentation updates.

Major Accomplishments:

- Benedicte:
    - Find the Thimble | Find the Red Thread (player, test)
- Dago
    - Even-Odd | Find the Red Thread (player, test)
- Joachim
    - Guess the number | Overall game of games (player, test)
    - UML Diagram, Method Glossary
- Myri
    - Coin-Flip | Overall game of games (player, test)
    - UML Diagram, Method Glossary
    - Responsibility Document

- Myri updated the tasks distribution in Monday.com
- Benedicte has been the point of contact with the implementation team, meanwhile Dago has been the point of contact with the design team
- Joachim and Myri started working on the overall Game of Games implementation where all individual games have been added
- The team agreed to keep working on separate branches to avoid mistakes and ambiguities

Next Steps:

Each member will individually complete the documentation related to each game they have been assigned. We will meet on Friday at 8 pm at Zoom to address any team needs and discuss the last steps before submission.

Meeting Notes:

In the meeting, we reviewed the current progress assignment, assigned Documentation to each member, and updated our Monday.com account for project management.

Documents

Testing_Document.docx [Tarek's group assignment]