

INDUSTRIAL TRAINING INTERNSHIP REPORT

INDUSTRIAL TRAINING SEMINAR REPORT TOPIC JAVA PROGRAMMING DOMAIN

Submitted in partial fulfilment of the degree of
B.Tech
Amrita School of Computing, Chennai



By

Shyam Tripathi

(CH.SC.U4CSE23160)

Amrita School of Computing, Chennai
Thrivallur, Tamilnadu

CERTIFICATE

This is to certify that Industrial Training Internship Report entitled **JAVA PROGRAMMING** has been submitted by Shyam Tripathi (**CH.SC.U4CSE23160**) for partial fulfilment of the Degree of B.Tech of Amrita School of Computing, Chennai . It is found satisfactory and approved for submission.

Date:

05/02/2024

Dr. S. Sountharajan

HOD,

Amrita School of Computing, Chennai

**Sampoojya Swami Vinayamritananda
Puri**

Director,

Amrita School of Computing, Chennai

OFFICIAL LETTER:



@internPE

INTERNSHIP COMPLETION CERTIFICATE

CID : IPIC#12345

To whomever it may concern

This is to certify that **SHYAM TRIPATHI** worked as an Intern in our company from **15-Jan-2024 to 11-Feb-2024**

Please find the internship details below:

Company Name: InternPe

Location: Remote

Domain : Java Programming

Designation: Intern

During their working period, we found him/her to be a sincere and dedicated intern with a professional attitude and very good knowledge of the job.

We thank him/her for their efforts and contribution and wish him/her the best in future endeavors.

Yours Sincerely

(Co-Founder)
InternPe



INTERNSHIP MENTOR DECLARATION

This is to certify that the Industrial Training Internship In **INTERNPE** entitled **JAVA PROGRAMMING** by Shyam Tripathi e has been done successfully and completed all the tasks provided in the internship.

Date:
05/02/2024



Ms. KRATI PORWAL
MENTOR,
INTERNPE
www.internpe.in



TASK DESCRIPTION

TASK 1:

Guess a Number Task:

OBJECTIVE:

1. Random Number Generation: Generate a random number between a specified range.
2. User Input: Allow the user to guess the generated number.
3. Comparison: Compare the user's guess with the generated number.
4. Feedback: Provide feedback indicating whether the guess is correct, too high, or too low.

HOW TO PERFORM:

1. Random Number Generation:
 - Use Java's `Random` class or another method to generate a random number within a specified range.
2. User Input:
 - Implement a mechanism to allow the user to input their guessed number.
3. Comparison:
 - Compare the user's guessed number with the randomly generated number to determine if it's correct, too high, or too low.
4. Feedback:
 - Provide clear feedback to the user based on the comparison, indicating whether the guess is correct or providing a hint for the next attempt.
5. Iteration (Optional):
 - Optionally, implement a loop to allow the user multiple attempts until they correctly guess the number or choose to exit the game.
6. User Interaction:
 - Ensure a user-friendly experience with clear instructions, and possibly add features like prompting the user to play again.
7. Scalability (Optional):
 - If desired, consider making the game scalable by allowing users to choose the range of numbers or introducing difficulty levels.

8. Testing:

- Test the game thoroughly to ensure that it works as expected, including scenarios where the user guesses correctly or incorrectly.

9. Documentation:

- Provide comments and documentation to explain the functionality of the code for future reference or collaboration.

TASK 2:

ROCK PAPER SCISSOR TASK:

OBJECTIVE:

1. User Input: Allow players to input their choice of rock, paper, or scissors.
2. Random Selection: Generate a random choice for the computer opponent.
3. Comparison: Compare user and computer choices to determine the winner based on the Rock-Paper-Scissors rules.
4. Feedback: Provide clear feedback indicating the winner or if it's a tie.

HOW TO PERFORM:

1. User Input:

- Allow players to input their choice of rock, paper, or scissors using user interaction mechanisms.

2. Random Selection:

- Generate a random choice for the computer opponent. Utilize Java's `Random` class or another method for randomness.

3. Comparison:

- Implement logic to compare the user's choice with the computer's choice based on the Rock-Paper-Scissors rules to determine the winner.

4. Feedback:

- Provide clear feedback to the user, indicating the winner or declaring a tie. This could be done through console output or a graphical user interface.

5. User Interaction:

- Ensure a user-friendly experience with clear instructions, and consider providing options for the user to play again or exit the game.

6. Scalability (Optional):

- Optionally, consider making the game scalable by adding more choices or introducing additional features, such as a scoring system.

7. Testing:

- Test the game thoroughly to ensure it works as expected under various scenarios, including winning, losing, and tying.

8. Documentation:

- Provide comments and documentation to explain the functionality of the code for future reference or collaboration.

TASK 3:

TIC TAC TOE TASK:

OBJECTIVE:

1. Grid Representation: Create a 3x3 grid to represent the Tic Tac Toe game board.
2. Player Turns: Allow two players to take turns marking spaces with 'X' and 'O'.
3. Winning Conditions: Define conditions for a player to win by having three marks in a row, column, or diagonal.
4. User Experience: Ensure a user-friendly experience with clear instructions, valid move validation, and an option to replay.

HOW TO PERFORM:

1. Grid Setup:
 - Design a 3x3 grid to represent the Tic Tac Toe game board.
2. Player Turns:
 - Allow two players to take turns marking spaces with 'X' and 'O'.
 - Implement a mechanism to switch between players after each move.
3. User Input:
 - Enable user interaction by allowing players to input their moves on the grid.
4. Winning Conditions:
 - Define winning conditions by checking for three marks in a row, column, or diagonal.
 - Implement logic to detect and announce a winner.
5. Tie Detection:
 - Implement logic to detect a tie when all spaces on the grid are filled and no player has won.
6. Feedback:
 - Provide clear feedback after each move, indicating the current player, the outcome of the move, and game status.
7. Reset and Replay:
 - Include functionality to reset the game board for a new match.
 - Allow players to replay without needing to restart the program.
8. Validation:

- Validate moves to ensure players can only mark empty spaces and avoid illegal moves.

9. User Interface Enhancements:

- Consider adding user-friendly features such as highlighting winning combinations or animating the mark placements.

10. Testing and Debugging:

- Test the game thoroughly to ensure smooth gameplay, including edge cases like invalid inputs, and debug any issues.

11. Optional Features (if time allows):

- Explore additional features like a scoreboard, customizable player marks, or a graphical user interface.

TASK 4:

CONNECT 4 GAME TASK:

OBJECTIVE:

1. Grid Structure: Create a 6x7 grid to represent the Connect 4 game board.
2. Player Interaction: Allow two players to take turns dropping colored discs into columns.
3. Winning Conditions: Define conditions for a player to win by connecting four discs horizontally, vertically, or diagonally.
4. User Experience: Ensure a user-friendly experience with clear instructions, valid move validation, and an option to replay.

HOW TO PERFORM:

Certainly! Here's a non-code, step-by-step guide on how to create a Connect 4 game in Java:

1. Grid Setup:

- Design a 6x7 grid to represent the Connect 4 game board.

2. Player Turns:

- Allow two players to take turns dropping colored discs into columns.
- Implement a mechanism to switch between players after each move.

3. Disc Placement:

- Enable user interaction by allowing players to drop their discs into columns.
- Implement logic to place the disc in the lowest available position within the selected column.

4. Winning Conditions:

- Define winning conditions by checking for four consecutive discs in a row, column, or diagonal.
- Implement logic to detect and announce a winner.

5. Tie Detection:

- Implement logic to detect a tie when the grid is full with no Connect 4.

6. Feedback:

- Provide clear feedback after each move, indicating the current player, the outcome of the move, and game status.

7. Reset and Replay:

- Include functionality to reset the game board for a new match.

- Allow players to replay without needing to restart the program.

8. Validation:

- Validate moves to ensure players can only drop discs in valid columns and avoid illegal moves.

9. User Interface Enhancements:

- Consider adding user-friendly features such as highlighting winning combinations or animating the disc placements.

10. Testing and Debugging:

- Test the game thoroughly to ensure smooth gameplay, including edge cases like invalid inputs, and debug any issues.

11. Optional Features (if time allows):

- Explore additional features like a scoreboard, customizable disc colors, or a graphical user interface.