**1. Tic Tac Toe Game**

**a. Project Name:** "Tic Tac Toe Challenge"

**b. Description:**

Develop a simple Tic Tac Toe game where two players can play against each other. The game should implement the logic for checking the winner and allow players to input their moves. The GUI is optional; you can create a console-based version first and add a graphical interface if desired.

**c. Features:**

* Game Logic: Implement logic to check for win conditions (three in a row, column, or diagonal) or a draw.
* Player Input: Allow players to input their moves (X or O) via the console or through a graphical interface.
* Game Loop: Continuously run the game until there is a winner or a draw.
* Optional GUI: If desired, create a graphical interface using Java Swing or JavaFX.

**d. Technologies Used:**

* Java: For programming the core logic and game mechanics.
* Java Swing/JavaFX (Optional): For building a graphical interface.

**e. Learning Objectives:**

* Learn to create a simple console-based or GUI-based game in Java.
* Understand game logic and how to implement it in Java.
* Improve problem-solving skills by handling different game scenarios.

SOLUTION:

1. Core Components of the Game Logic:

Game Board: Create a 3x3 grid, either using a 2D array or a list, where players will make their moves.

Win Conditions: Implement methods to check rows, columns, and diagonals for three identical marks (either "X" or "O") for a win condition.

Draw Condition: Check if the grid is fully occupied without a winner, declaring a draw.

2. Player Input:

Console Input: Use the Scanner class to read player input (position on the grid for the move) for a basic console-based version.

Input Validation: Ensure each move is within the grid range and that players don’t overwrite each other's moves.

3. Game Loop:

Set up a loop that continues to ask for player moves, updates the board, and checks for a winner or draw after each move.

Alternate between the two players, keeping track of whose turn it is.

4. Optional GUI (Java Swing/JavaFX):

Use Java Swing or JavaFX to create a simple grid layout that represents the Tic Tac Toe board.

Add buttons for each cell in the grid and update them based on player moves.

Include visual cues for a win (highlighting winning row/column/diagonal) or a draw message.

5. Technologies Used:

Java: For all core game logic and mechanics.

Java Swing/JavaFX (optional): To make the game visually interactive.

6. Learning Objectives:

Develop basic game logic for a well-known game.

Practice coding a loop that handles user input and logic checks in Java.

Optionally, gain experience in building GUIs, which can be useful for more complex applications

CODE:

import java.util.Scanner;

public class TicTacToeChallenge {

private static char[][] board = { { '1', '2', '3' }, { '4', '5', '6' }, { '7', '8', '9' } };

private static char currentPlayer = 'X';

public static void main(String[] args) {

boolean gameRunning = true;

Scanner scanner = new Scanner(System.in);

while (gameRunning) {

printBoard();

System.out.println("Player " + currentPlayer + ", enter a number (1-9) to place your mark:");

int move = scanner.nextInt();

if (placeMark(move)) {

if (checkWin()) {

printBoard();

System.out.println("Player " + currentPlayer + " wins!");

gameRunning = false;

} else if (isBoardFull()) {

printBoard();

System.out.println("The game is a draw!");

gameRunning = false;

} else {

switchPlayer();

}

} else {

System.out.println("Invalid move, try again.");

}

}

scanner.close();

}

private static void printBoard() {

for (char[] row : board) {

for (char cell : row) {

System.out.print(cell + " ");

}

System.out.println();

}

}

private static boolean placeMark(int move) {

int row = (move - 1) / 3;

int col = (move - 1) % 3;

if (move >= 1 && move <= 9 && board[row][col] != 'X' && board[row][col] != 'O') {

board[row][col] = currentPlayer;

return true;

}

return false;

}

private static void switchPlayer() {

currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';

}

private static boolean checkWin() {

// Check rows, columns, and diagonals

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

if (board[i][0] == currentPlayer && board[i][1] == currentPlayer && board[i][2] == currentPlayer)

return true;

if (board[0][i] == currentPlayer && board[1][i] == currentPlayer && board[2][i] == currentPlayer)

return true;

}

if (board[0][0] == currentPlayer && board[1][1] == currentPlayer && board[2][2] == currentPlayer)

return true;

if (board[0][2] == currentPlayer && board[1][1] == currentPlayer && board[2][0] == currentPlayer)

return true;

return false;

}

private static boolean isBoardFull() {

for (char[] row : board) {

for (char cell : row) {

if (cell != 'X' && cell != 'O') {

return false;

}

}

}

return true;

}

}

EXPLANATION:

.

1.The board array represents the 3x3 Tic Tac Toe grid.

2.The currentPlayer keeps track of whether it’s player 'X' or 'O's turn.

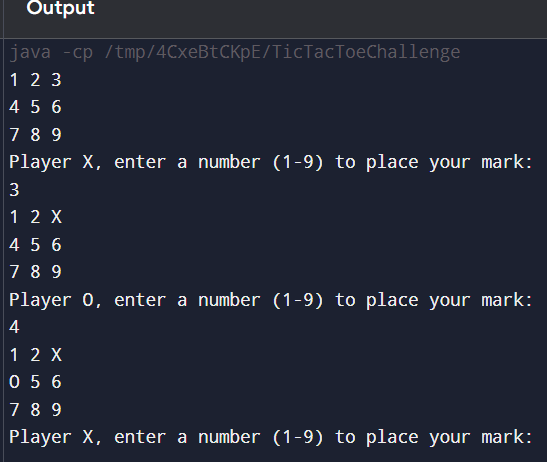
3.The printBoard method displays the board to the console.

4.The placeMark method updates the board if the chosen cell is valid.

5.The checkWin method checks for win conditions in rows, columns, and diagonals.

6.The isBoardFull method checks if the board is completely filled.

OUTPUT:



**2. Calculator**

**a. Project Name** "Smart Calculator"

**b. Description:**

Build a basic calculator application that can perform simple arithmetic operations like addition, subtraction, multiplication, and division. Start with a console-based version and optionally add a graphical interface with buttons for digits and operations.

**c. Features:**

* Basic Operations: Implement functionality for basic arithmetic operations.
* User Input: Allow user input via the console or GUI.
* Clear and Backspace: Implement functionality to clear the current input or delete the last digit entered.
* Error Handling: Handle errors such as division by zero and invalid inputs.
* Optional GUI: If desired, create a graphical interface using Java Swing or JavaFX with buttons for numbers and operations.

**d. Technologies Used:**

* Java: For programming the core functionality.
* Java Swing/JavaFX (Optional): For building a graphical interface.

**e. Learning Objectives:**

* Learn to create a functional application in Java.
* Practice handling user input and implementing basic arithmetic logic. - Understand error handling and validation in Java applications.

SOLUTION:

CODE EXPLANATION:

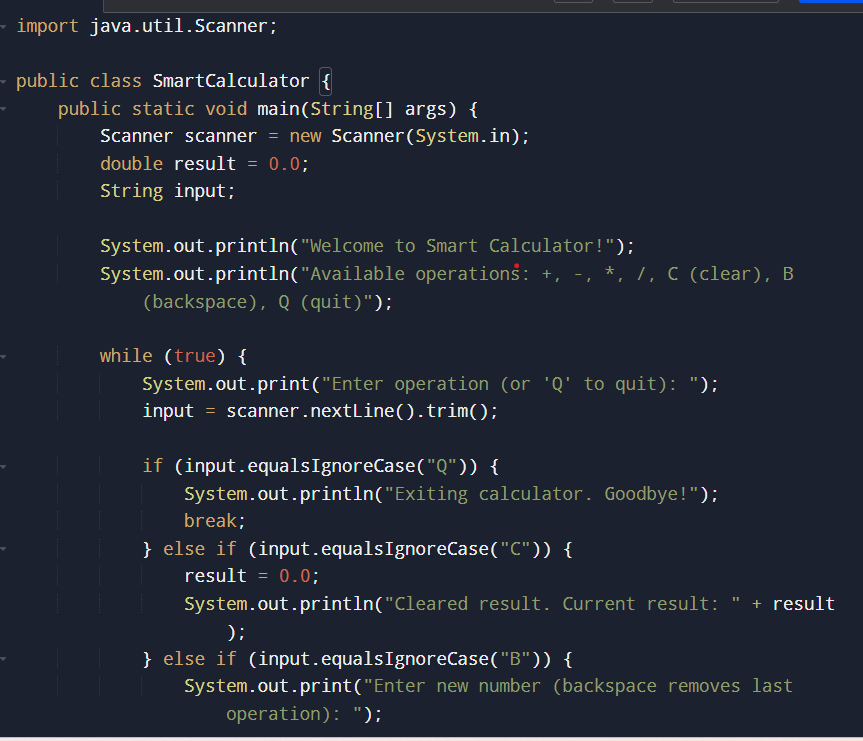
1.User Input: Allows the user to input the arithmetic operation (+, -, \*, /, C to clear, B for backspace, and Q to quit).

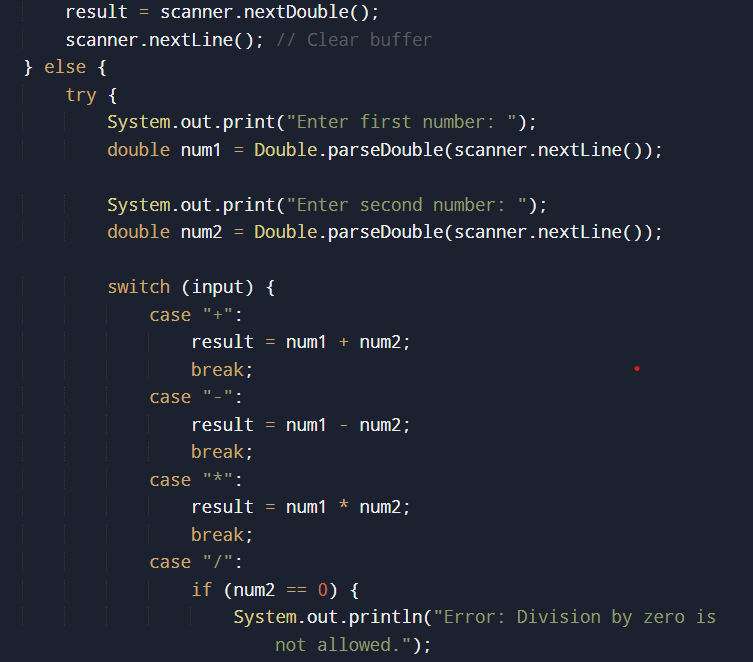
2.Operations: The program performs the selected operation, updating and displaying the result.

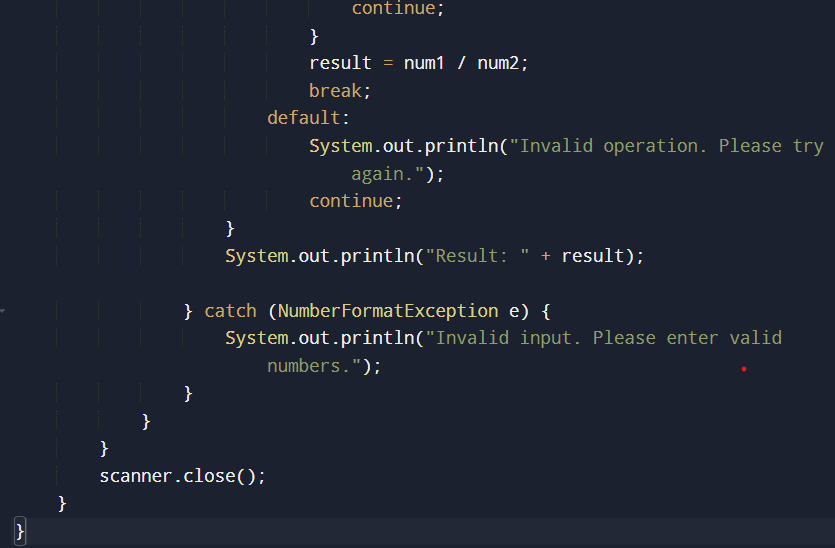
3.Error Handling:

a.Division by Zero: Checks if the denominator is zero before performing division.

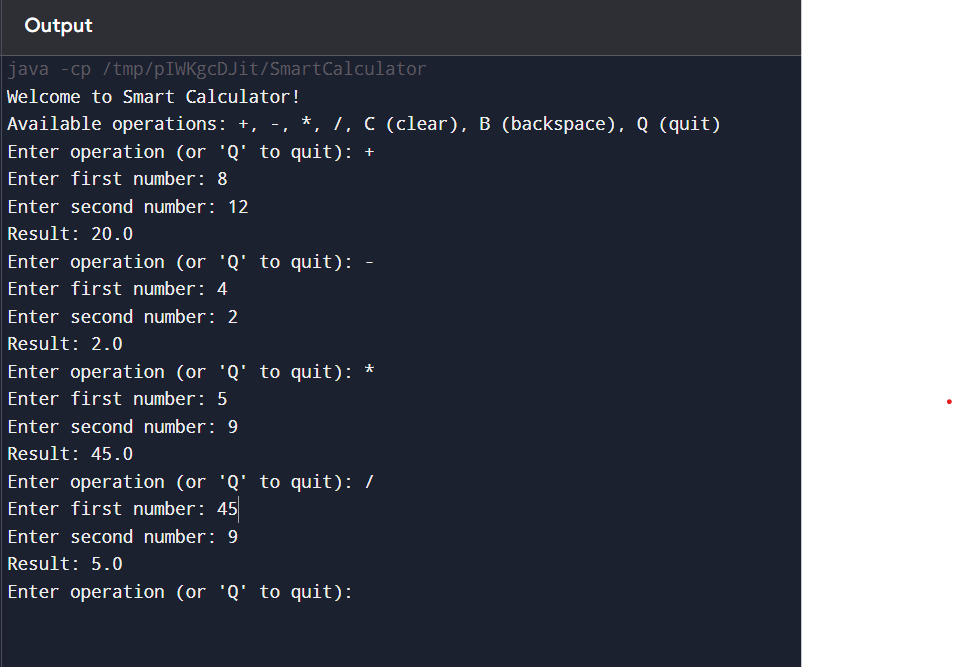
b.Invalid Input: Catches NumberFormatException to handle non-numeric inputs gracefully.

CODE:  






OUTPUT:



**3. ExpenseTracker**

**a. Project Name:** "Personal Expense Tracker"

**b. Description:**

Develop an expense tracker application to help users manage their daily expenses. The application should allow users to add, view, and delete expenses, and categorize them for better management. Start with a console-based version and optionally add a GUI.

**c. Features:**

* Add, View, and Delete Expenses: Allow users to manage their expenses through commands.
* Expense Categories: Ability to categorize expenses (e.g., Food, Transport, Entertainment).
* Expense Summary: Display a summary of expenses by category and total spending.
* Data Persistence: Save expenses to a file or database and load them on startup.
* Optional GUI: If desired, provide a graphical interface for managing expenses using Java Swing or JavaFX.

**d. Technologies Used:**

* Java: For the core logic and application structure.
* File Handling/Database (Optional): To store and retrieve expenses. - Java Swing/JavaFX (Optional): For building a graphical interface.

**e. Learning Objectives:**

* Learn to develop a Java application that handles user input and displays data.
* Understand file handling or database connectivity in Java.
* Gain experience in creating a user-friendly interface and managing data persistence.

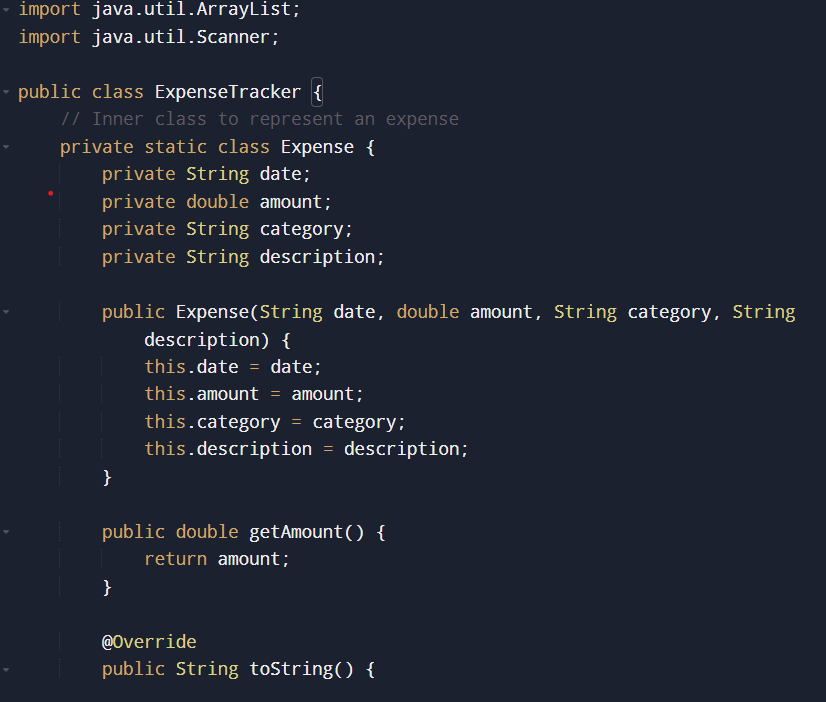
SOLUTION:

* EXPLANATION FOR CODE :

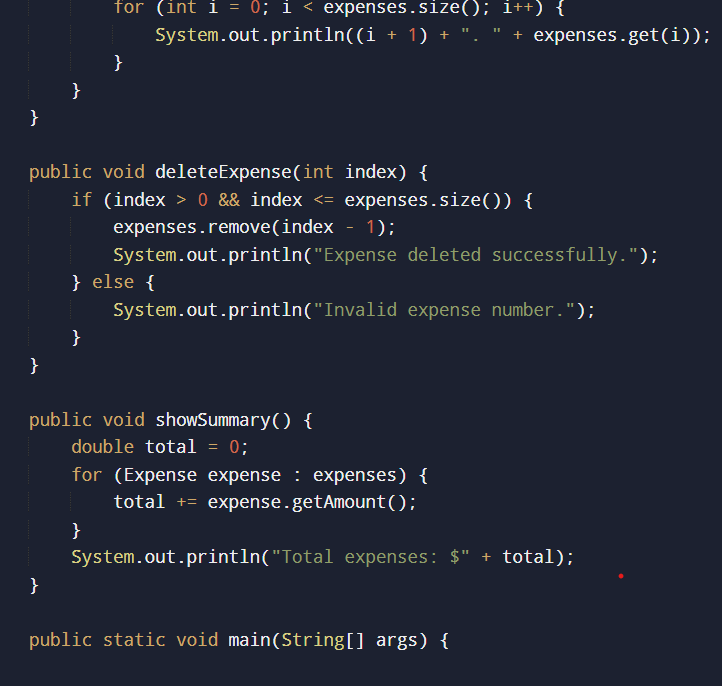
1. Inner Class: The Expense class is now defined inside the ExpenseTracker class to keep everything in one file.

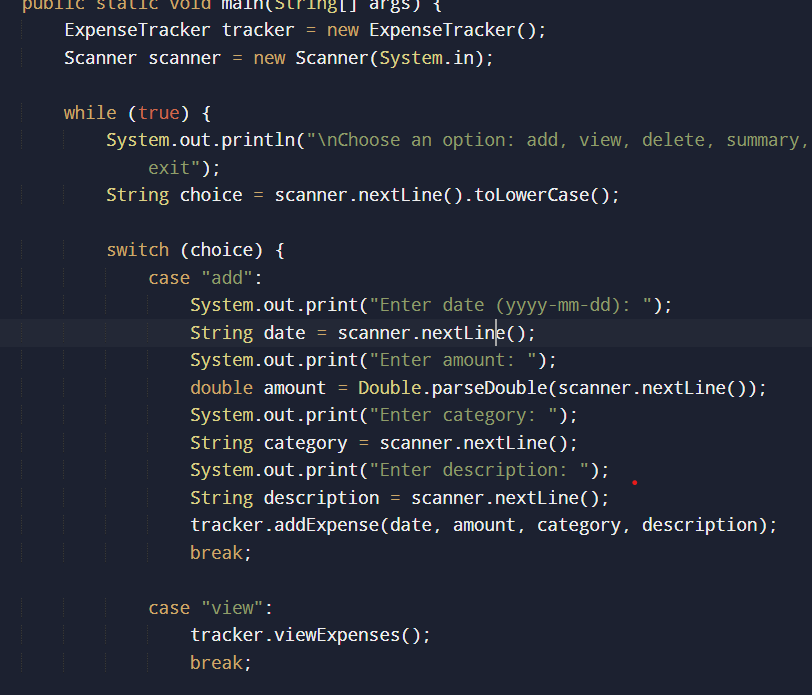
2. Main Program Logic: The main method contains a loop to handle user commands (add, view, delete, summary, and exit).

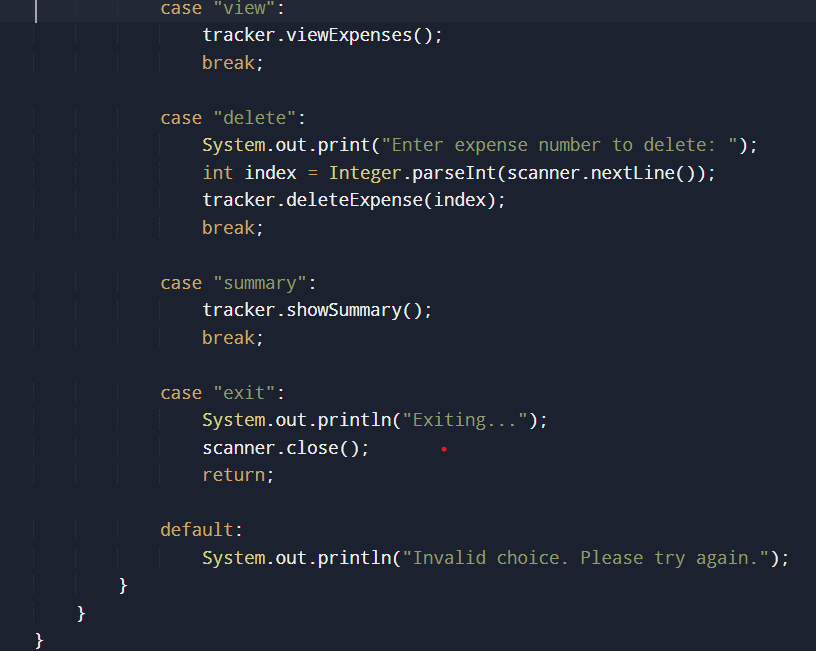
CODE:



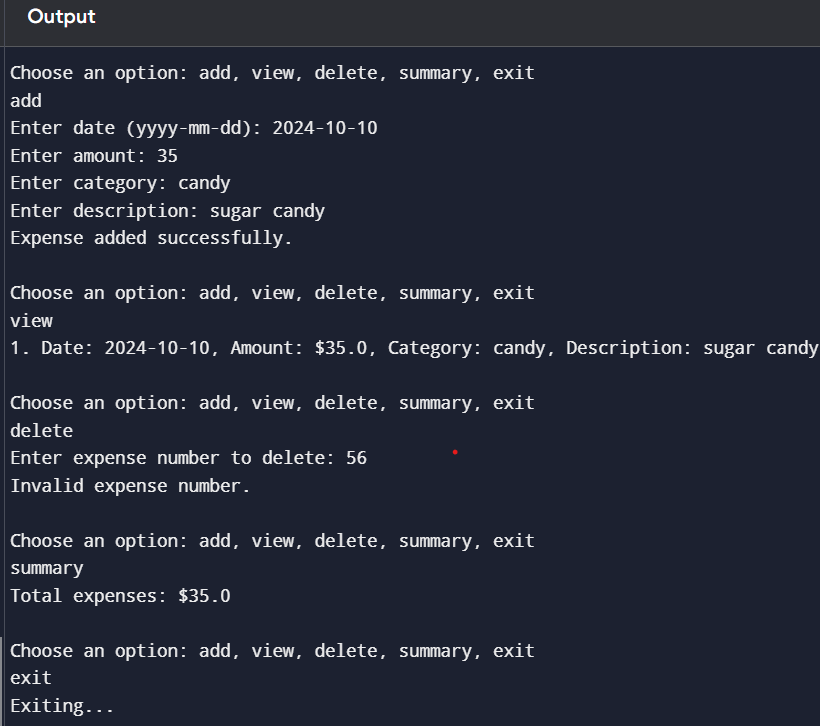








OUTPUT:



**4. Link Shortener**

**a. Project Name:** "QuickLink Shortener"

**b. Description:**

Create a simple link shortener application that takes a long URL and generates a shorter version of it. The application should store the original and shortened URLs and provide a way to retrieve the original URL using the shortened version. Start with a console-based version and optionally add a GUI.

**c. Features:**

* URL Input and Output: Accept a long URL and provide a shortened version.
* URL Mapping: Generate a unique short link for each long URL.
* Data Storage: Store the original and shortened URLs in a file or database.
* URL Redirection Simulation: Simulate redirecting the user to the original URL when accessing the shortened link.
* Error Handling: Handle invalid URLs and duplicate entries gracefully.
* Optional GUI: If desired, create a graphical interface for entering URLs and displaying results using Java Swing or JavaFX.

**d. Technologies Used:**

* Java: For core functionality and URL mapping logic.
* File Handling/Database (Optional): To store and manage URL mappings.
* Java Swing/JavaFX (Optional): For building a graphical interface.

**e. Learning Objectives:**

* Learn to develop a Java application that processes user input and manages data.
* Understand URL handling and management in a Java context.
* Practice working with data storage and retrieval using files or databases.

SOLUTION:

EXPLANATON :

1. HashMap: urlMap stores the mapping between the shortened URL and the original URL.

2. shortenURL: Generates a short URL by converting the hash of the original URL to a hexadecimal string.

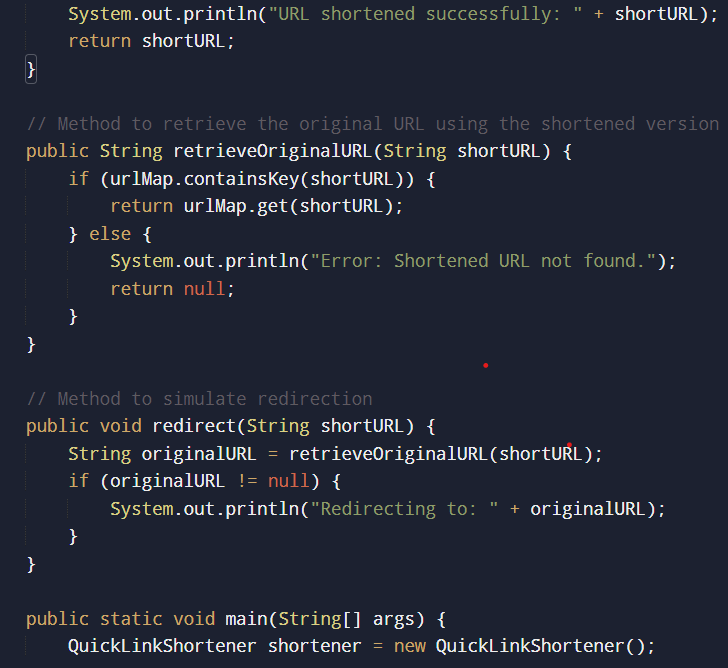
If a URL has already been shortened, it retrieves and returns the existing short URL.

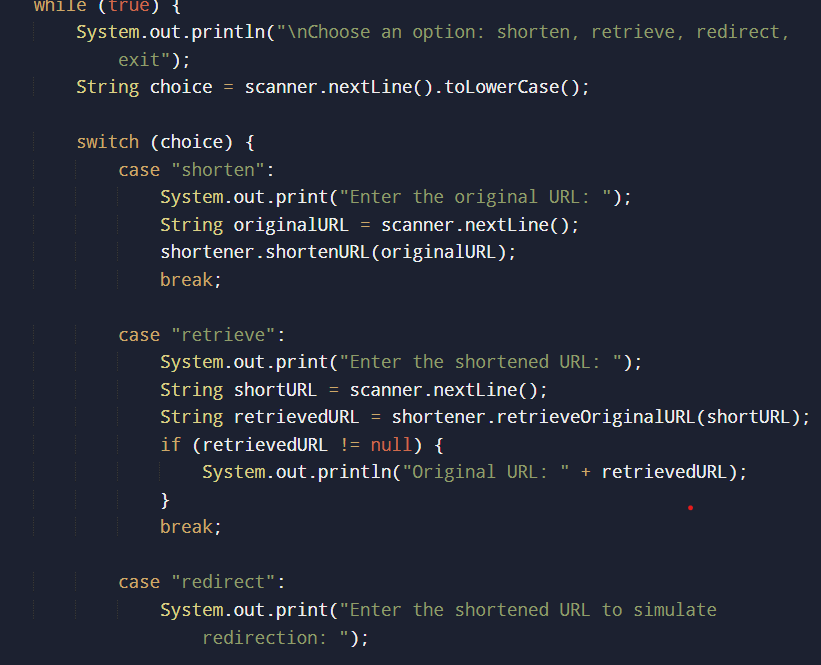
3. retrieveOriginalURL: Looks up the original URL using the shortened URL.

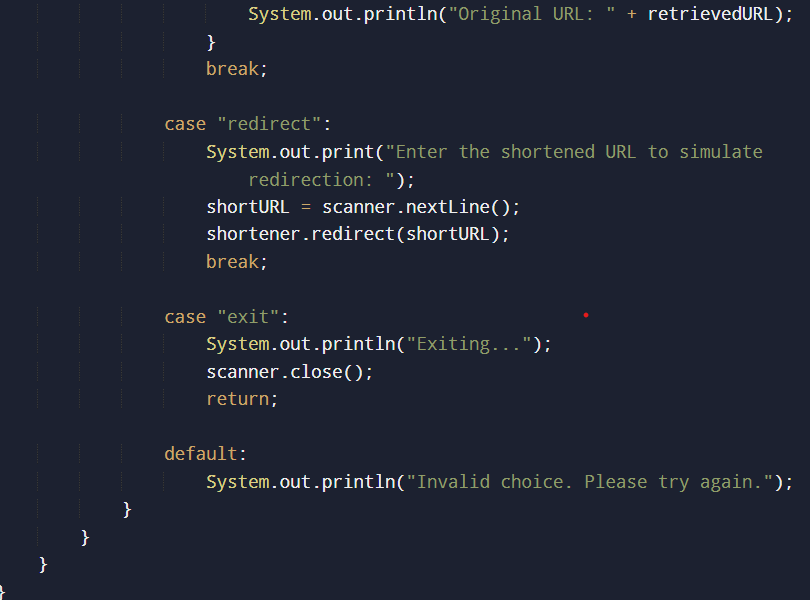
4. redirect: Simulates redirection by retrieving the original URL and printing it.

CODE:









OUTPUT:

