CS 602 Java, HW#1 – Group H(Team Work)

Group Members:

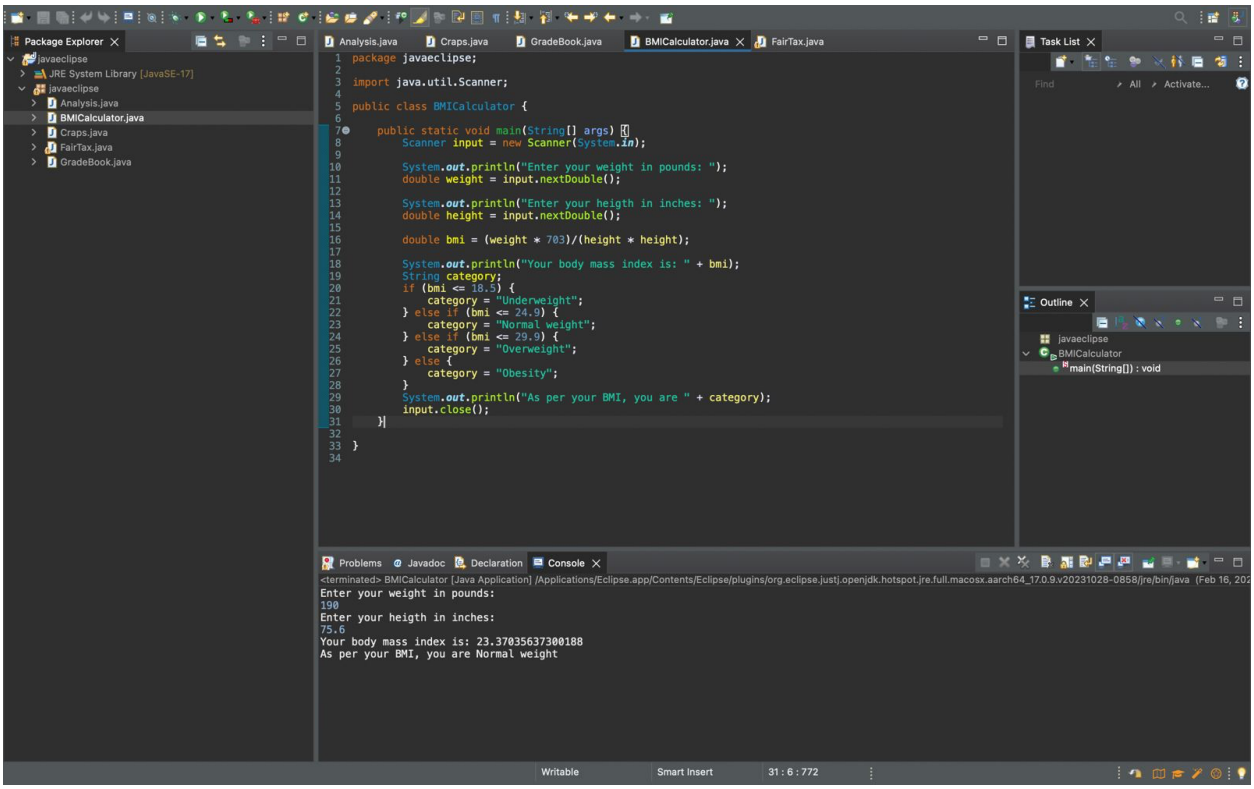
- Charitha Kammari(ck369)
- Mohammed Hannan Desai(mmd76)
- Joy Patel(jp2267)
- Dhruval Dhameliya(dd548)
- Revanth Guntupalli(rg757)

3. Use Eclipse, create, compile, run, print (program listing and results) and explain:

a. How 2.33 works, pages 66-67 - Team (1 point)

Description:

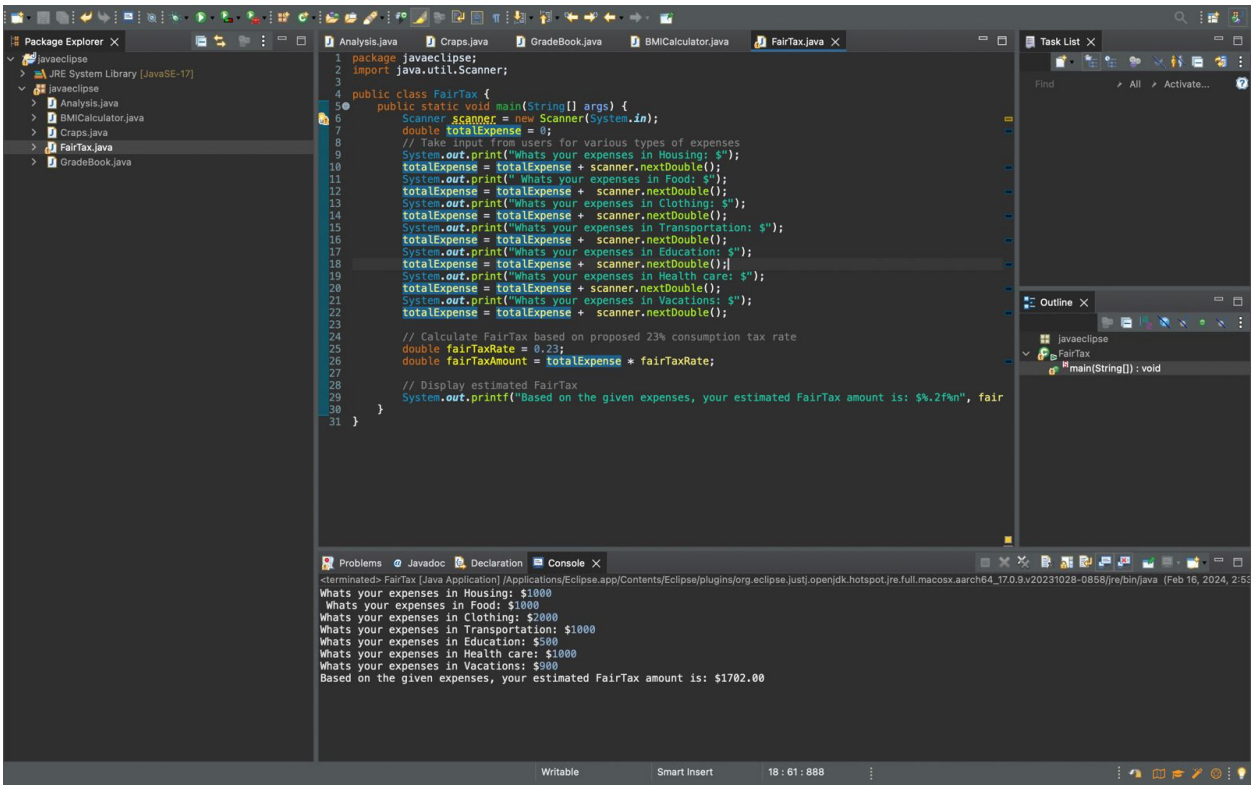
This code acts as a personal BMI calculator! It first opens a channel to receive your weight (in pounds) and height (in inches) through prompts. Using these values, it calculates your Body Mass Index (BMI) using a specific formula. Based on the calculated BMI, it then categorizes you as "Underweight", "Normal weight", "Overweight", or "Obese" based on established ranges. Finally, it presents both your BMI value and the corresponding category, giving you a quick snapshot of your body mass index.



c.How 4.31 works on page 151 - Team (1 point)

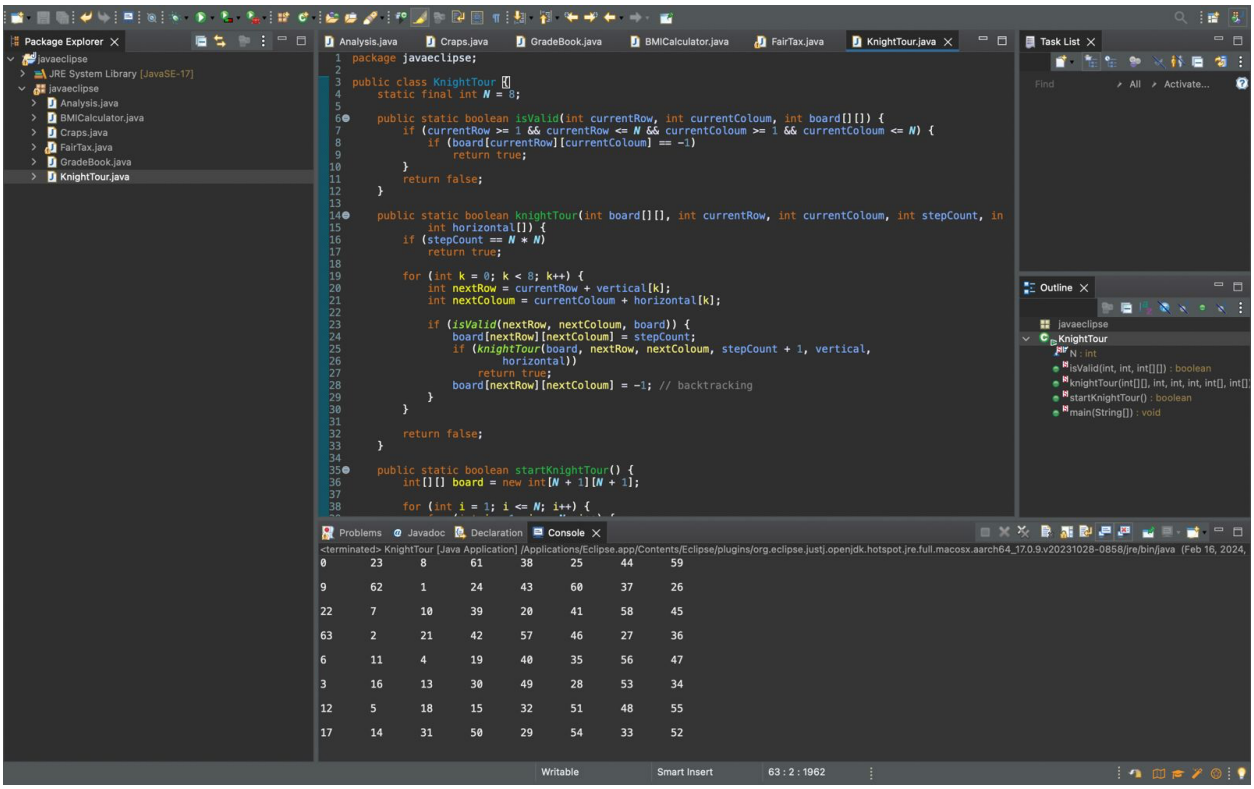
Description: This code acts as a FairTax calculator! It gathers your expenses across various categories like housing, food, and healthcare through console prompts. Once you input your spending in each category, the code sums them up to calculate your total expenses. The core of the calculation lies in the FairTax rate. The code assumes a proposed 23% consumption tax and multiplies your total expenses by this rate to determine your estimated FairTax amount. Finally, it presents you with the calculated FairTax figure, offering an estimate of what you might pay under the FairTax proposal.

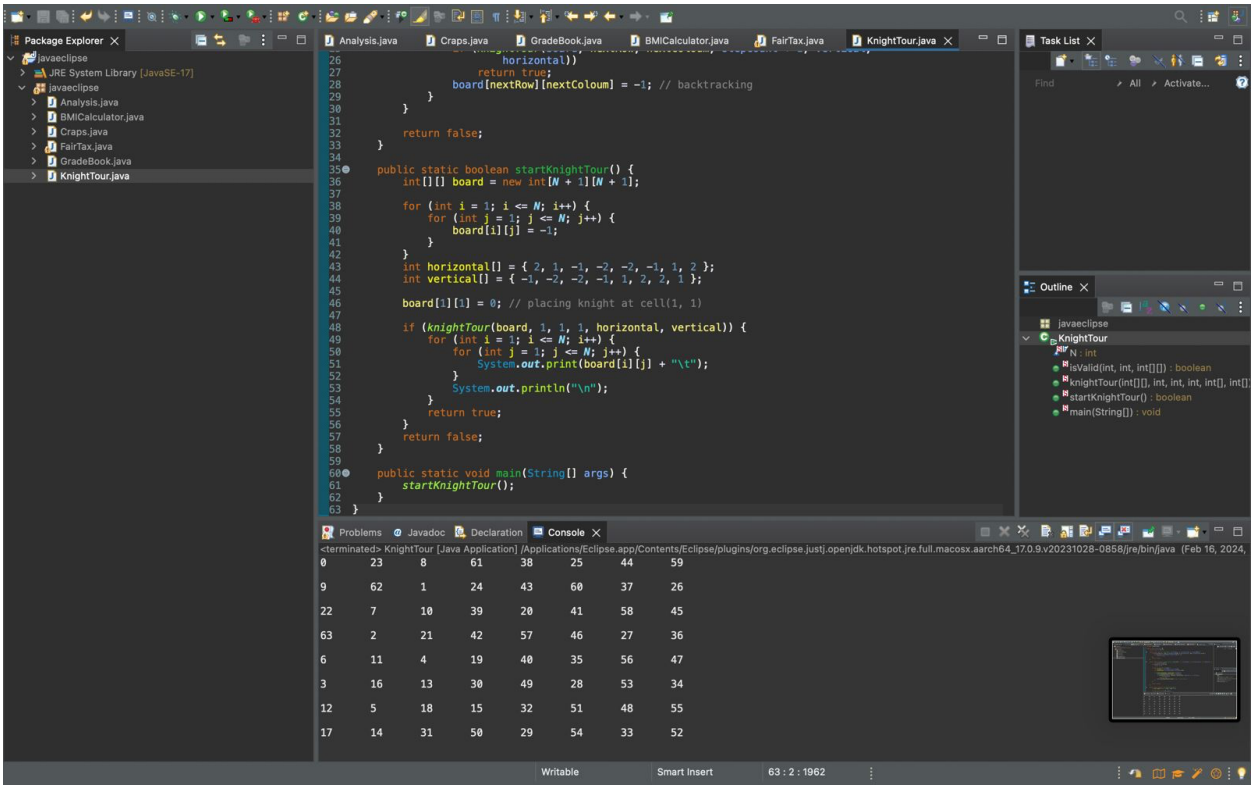
Code:



e. How 6.22 works, pages 235 - 237 – Team (2 points)

Description: This code addresses the classic "Knight's Tour" problem on an 8x8 chessboard, seeking to find a sequence of moves for a knight starting at (1,1) that covers every square exactly once without revisiting any. It initializes a board represented as a 2D array, initially filled with -1s to denote empty squares, and includes a validity check function to ensure moves stay within the board and land on empty squares. The core function, knightTour, recursively explores all possible knight moves, backtracking when necessary. The startKnightTour function initializes the board, places the knight at the starting position, and begins the search. If a solution is found, it prints the tour sequence; otherwise, it returns False. In essence, the code uses backtracking to systematically explore potential moves until a complete tour is discovered, presenting the successful sequence visually on the board.
Code:





g. How 8.18 works, pages 325 -326 – Team (1 point)

Description: This program breathes new life into the classic game of tic-tac-toe through a well-structured code design. It begins by defining an 'enum' called 'Cell' to distinguish between empty squares, Xs, and Os. A 3x3 board array is then set up, initially filled with empty cells. A boolean variable 'playerXTurn' keeps track of which player starts the game. The 'playGame' method oversees the game's progression, welcoming players, managing turns until a win or draw is reached, validating moves, and updating the board accordingly. It alternates turns between players, continuously checking for wins or draws after each move, and handles invalid moves by prompting for new ones. The current state of the board is displayed using 'printBoard', while 'isValidMove' and 'makeMove' functions ensure that moves are valid and executed properly. 'checkWin' and 'checkDraw' functions determine if the game has reached a conclusion. Finally, the main method initializes a TicTacToe game and calls 'playGame' to kick off the gameplay. This concise implementation encapsulates the essence of tic-tac-toe, offering user interaction, move validation, and clear display of progress and outcomes.

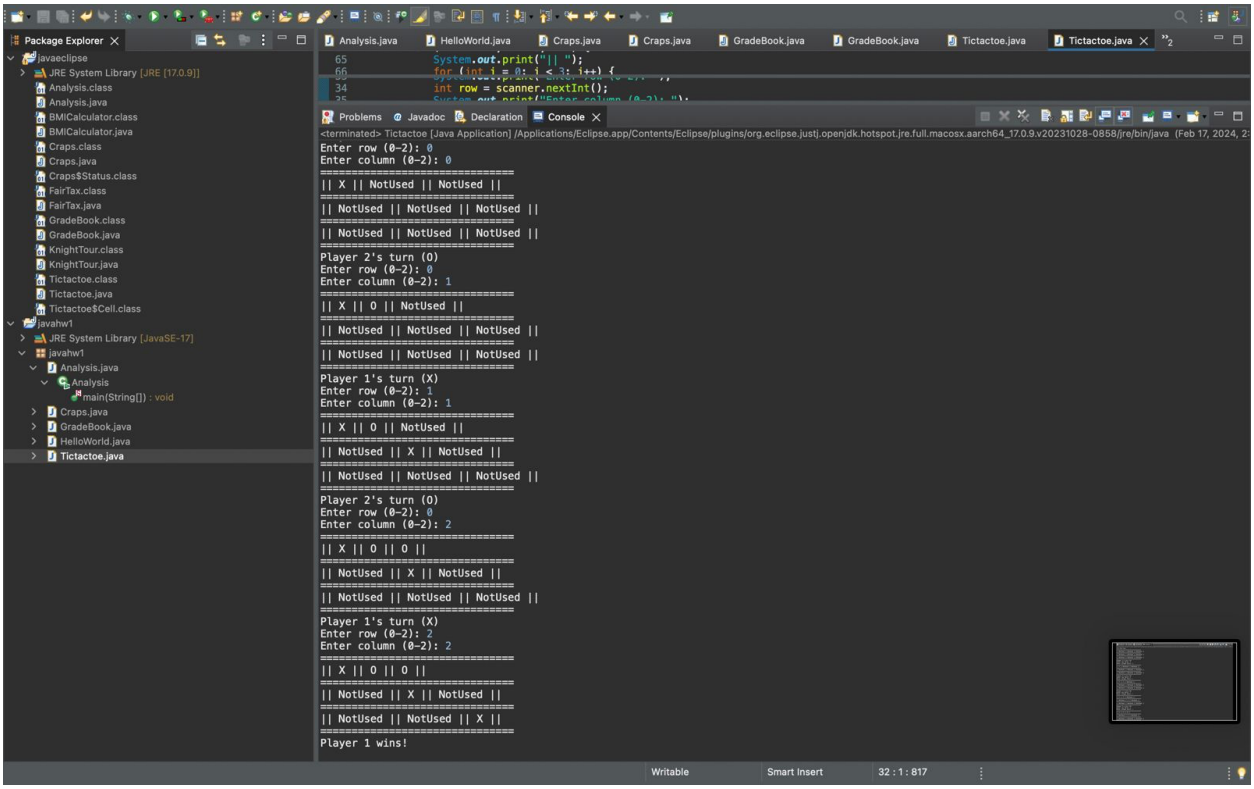
Code:

```java
package javahw1;
import java.util.Scanner;

public class Tictactoe {
    private enum Cell { X, O, NotUsed }
    private Cell[][] board;
    private boolean playerXTurn;

    public Tictactoe() {
        board = new Cell[3][3];
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                board[i][j] = Cell.NotUsed;
            }
        }
        playerXTurn = true;
    }

    public void startGame() {
        Scanner scanner = new Scanner(System.in);
        boolean gameFinished = false;

        System.out.println("Tic-Tac-Toe!");

        while (!gameFinished) {
            printBoard();
            if (playerXTurn) {
                System.out.println("Player 1's turn (X)");
            } else {
                System.out.println("Player 2's turn (O)");
            }

            System.out.print("Enter row (0-2): ");
            int row = scanner.nextInt();
            System.out.print("Enter column (0-2): ");
            int col = scanner.nextInt();

            if (isValidMove(row, col)) {
                makeMove(row, col);
                if (isWon()) {
                    printBoard();
                    if (playerXTurn) {
                        System.out.println("Player 1 wins!");
                    } else {
                        System.out.println("Player 2 wins!");
                    }
                    gameFinished = true;
                } else if (isDraw()) {
                    printBoard();
                    System.out.println("Tie Game!");
                    gameFinished = true;
                } else {
```

```
    public void startGame() {
        Scanner scanner = new Scanner(System.in);
        boolean gameFinished = false;

        System.out.println("Tic-Tac-Toe!");

        while (!gameFinished) {
            printBoard();
            if (playerXTurn) {
                System.out.println("Player 1's turn (X)");
            } else {
                System.out.println("Player 2's turn (O)");
            }

            System.out.print("Enter row (0-2): ");
            int row = scanner.nextInt();
            System.out.print("Enter column (0-2): ");
            int col = scanner.nextInt();

            if (isValidMove(row, col)) {
                makeMove(row, col);
                if (isWon()) {
                    printBoard();
                    if (playerXTurn) {
                        System.out.println("Player 1 wins!");
                    } else {
                        System.out.println("Player 2 wins!");
                    }
                    gameFinished = true;
                } else if (isDraw()) {
                    printBoard();
                    System.out.println("Tie Game!");
                    gameFinished = true;
                } else {
                    playerXTurn = !playerXTurn;
                }
            } else {
                System.out.println("Incorrect Move!");
            }
        }
        scanner.close();
    }

    private void printBoard() {
        System.out.println("=============================");
        for (int i = 0; i < 3; i++) {
            System.out.print("|| ");
            for (int j = 0; j < 3; j++) {
                System.out.print(board[i][j] + " || ");
            }
            System.out.println();
            System.out.println("=============================");
        }
    }

    private boolean isValidMove(int row, int col) {
        return row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] == Cell.NotUsed;
    }

    private void makeMove(int row, int col) {
        if (playerXTurn) {
            board[row][col] = Cell.X;
        } else {
            board[row][col] = Cell.O;
        }
    }

    private boolean isWon() {
        for (int i = 0; i < 3; i++) {
            if (board[i][0] != Cell.NotUsed && board[i][0] == board[i][1] && board[i][0] == board[i][2]) {
                return true;
            }
            if (board[0][i] != Cell.NotUsed && board[0][i] == board[1][i] && board[0][i] == board[2][i]) {
                return true;
            }
        }
        if (board[0][0] != Cell.NotUsed && board[0][0] == board[1][1] && board[0][0] == board[2][2]) {
            return true;
        }
        return board[0][2] != Cell.NotUsed && board[0][2] == board[1][1] && board[0][2] == board[2][0];
    }

    private boolean isDraw() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i][j] == Cell.NotUsed) {
                    return false;
                }
            }
        }
        return true;
    }

    public static void main(String[] args) {
```

```
                    return false;
                }
            }
        }
        return true;
    }

    public static void main(String[] args) {
        Tictactoe game = new Tictactoe();
        game.startGame();
    }
}
```

Output:




```
Player 1's turn (X)
Enter row (0-2): 2
Enter column (0-2): 2
|| X || O || O ||
|| NotUsed || X || NotUsed ||
|| NotUsed || NotUsed || X ||
Player 1 wins!
```