

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.

```
1 package javacclipse;
2
3 import java.util.Scanner;
4
5 public class BMICalculator {
6
7     public static void main(String[] args) {
8         Scanner input = new Scanner(System.in);
9
10        System.out.println("Enter your weight in pounds: ");
11        double weight = input.nextDouble();
12
13        System.out.println("Enter your height in inches: ");
14        double height = input.nextDouble();
15
16        double bmi = (weight * 703)/(height * height);
17
18        System.out.println("Your body mass index is: " + bmi);
19        String category;
20        if (bmi <= 18.5) {
21            category = "Underweight";
22        } else if (bmi <= 24.9) {
23            category = "Normal weight";
24        } else if (bmi <= 29.9) {
25            category = "Overweight";
26        } else {
27            category = "Obesity";
28        }
29        System.out.println("As per your BMI, you are " + category);
30        input.close();
31    }
32
33 }
34
```

```
190
Enter your weight in pounds:
75.6
Enter your height in inches:
Your body mass index is: 23.37035637300188
As per your BMI, you are Normal weight
```

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:

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure with files Analysis.java, Craps.java, GradeBook.java, BMICalculator.java, and FairTax.java.
- Editor:** Displays the FairTax.java code. The code prompts the user for various types of expenses (Housing, Food, Clothing, Transportation, Education, Health care, Vacations) and calculates a FairTax amount based on a proposed 23% consumption tax rate.
- Console:** Shows the execution output of the program. The user inputs the following expenses:
 - Whats your expenses in Housing: \$1000
 - Whats your expenses in Food: \$1000
 - Whats your expenses in Clothing: \$2000
 - Whats your expenses in Transportation: \$1000
 - Whats your expenses in Education: \$500
 - Whats your expenses in Health care: \$1000
 - Whats your expenses in Vacations: \$900
 The program then outputs: "Based on the given expenses, your estimated FairTax amount is: \$1702.00"

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:

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure under "javaclipse".
- Code Editor:** Displays the `KnightTour.java` file. The code implements a backtracking algorithm to solve the N-Queens problem. It includes methods for validating a move, performing a knight tour, starting the tour, and printing the board state.
- Task List:** Shows a single task named "N: int".
- Outline:** Lists the class members: `isValid(int, int, int[][], boolean)`, `knightTour(int[][], int, int, int, int[], int[], boolean)`, `startKnightTour() : boolean`, and `main(String[]) : void`.
- Console:** Shows the output of the program, which is a 8x8 chessboard representation where each number represents a knight's move sequence from (1,1) to (8,8).

```

1 package javaclipse;
2
3 public class KnightTour {
4     static final int N = 8;
5
6     public static boolean isValid(int currentRow, int currentColoum, int board[][]) {
7         if (currentRow >= 1 && currentRow <= N && currentColoum >= 1 && currentColoum <= N) {
8             if (board[currentRow][currentColoum] == -1)
9                 return true;
10        }
11        return false;
12    }
13
14    public static boolean knightTour(int board[][], int currentRow, int currentColoum, int stepCount, int horizontal[], int vertical[], int nextRow[], int nextColoum[], int N) {
15        if (stepCount == N * N)
16            return true;
17
18        for (int k = 0; k < 8; k++) {
19            int nextRow = currentRow + vertical[k];
20            int nextColoum = currentColoum + horizontal[k];
21
22            if (isValid(nextRow, nextColoum, board)) {
23                board[nextRow][nextColoum] = stepCount;
24                if (knightTour(board, nextRow, nextColoum, stepCount + 1, vertical,
25                               horizontal))
26                    return true;
27                board[nextRow][nextColoum] = -1; // backtracking
28            }
29        }
30
31        return false;
32    }
33
34    public static boolean startKnightTour() {
35        int[][] board = new int[N + 1][N + 1];
36
37        for (int i = 1; i <= N; i++) {
38            for (int j = 1; j <= N; j++) {
39                board[i][j] = -1;
40            }
41        }
42
43        int horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
44        int vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
45
46        board[1][1] = 0; // placing knight at cell(1,1)
47
48        if (knightTour(board, 1, 1, 1, horizontal, vertical))
49            for (int i = 1; i <= N; i++) {
50                for (int j = 1; j <= N; j++) {
51                    System.out.print(board[i][j] + "\t");
52                }
53                System.out.println("\n");
54            }
55        return true;
56    }
57
58    public static void main(String[] args) {
59        startKnightTour();
60    }
61
62 }
63

```

This screenshot is nearly identical to the one above, showing the same code editor, console output, and Eclipse interface. The only difference is in the Outline view, which now includes the `isValid(int, int, int[][], boolean)` method.

```

1 package javaclipse;
2
3 public class KnightTour {
4     static final int N = 8;
5
6     public static boolean isValid(int currentRow, int currentColoum, int board[][], boolean) {
7         if (currentRow >= 1 && currentRow <= N && currentColoum >= 1 && currentColoum <= N) {
8             if (board[currentRow][currentColoum] == -1)
9                 return true;
10        }
11        return false;
12    }
13
14    public static boolean knightTour(int board[][], int currentRow, int currentColoum, int stepCount, int horizontal[], int vertical[], int nextRow[], int nextColoum[], int N) {
15        if (stepCount == N * N)
16            return true;
17
18        for (int k = 0; k < 8; k++) {
19            int nextRow = currentRow + vertical[k];
20            int nextColoum = currentColoum + horizontal[k];
21
22            if (isValid(nextRow, nextColoum, board, false)) {
23                board[nextRow][nextColoum] = stepCount;
24                if (knightTour(board, nextRow, nextColoum, stepCount + 1, vertical,
25                               horizontal))
26                    return true;
27                board[nextRow][nextColoum] = -1; // backtracking
28            }
29        }
30
31        return false;
32    }
33
34    public static boolean startKnightTour() {
35        int[][] board = new int[N + 1][N + 1];
36
37        for (int i = 1; i <= N; i++) {
38            for (int j = 1; j <= N; j++) {
39                board[i][j] = -1;
40            }
41        }
42
43        int horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
44        int vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
45
46        board[1][1] = 0; // placing knight at cell(1,1)
47
48        if (knightTour(board, 1, 1, 1, horizontal, vertical))
49            for (int i = 1; i <= N; i++) {
50                for (int j = 1; j <= N; j++) {
51                    System.out.print(board[i][j] + "\t");
52                }
53                System.out.println("\n");
54            }
55        return true;
56    }
57
58    public static void main(String[] args) {
59        startKnightTour();
60    }
61
62 }
63

```

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:

```

18● public void startGame() {
19    Scanner scanner = new Scanner(System.in);
20    boolean gameFinished = false;
21
22    System.out.println("Tic-Tac-Toe!");
23
24    while (!gameFinished) {
25        printBoard();
26        if (playerXTurn) {
27            System.out.println("Player 1's turn (X)");
28        } else {
29            System.out.println("Player 2's turn (O)");
30        }
31
32        System.out.print("Enter row (0-2): ");
33        int row = scanner.nextInt();
34        System.out.print("Enter column (0-2): ");
35        int col = scanner.nextInt();
36
37        if (isValidMove(row, col)) {
38            makeMove(row, col);
39            if (isWon()) {
40                printBoard();
41                if (playerXTurn) {
42                    System.out.println("Player 1 wins!");
43                } else {
44                    System.out.println("Player 2 wins!");
45                }
46                gameFinished = true;
47            } else if (isDraw()) {
48                printBoard();
49                System.out.println("Tie Game!");
50                gameFinished = true;
51            } else {
52                playerXTurn = !playerXTurn;
53            }
54        } else {
55            System.out.println("Incorrect Move!");
56        }
57    }
58    scanner.close();
59}
60
61● private void printBoard() {
62    System.out.println("=====");
63    for (int i = 0; i < 3; i++) {
64        System.out.print("|| ");
65        for (int j = 0; j < 3; j++) {
66            System.out.print(board[i][j] + " || ");
67        }
68        System.out.println();
69        System.out.print("=====\n");
70    }
71}
72
73● private boolean isValidMove(int row, int col) {
74    return row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] == Cell.NotUsed;
75}
76
77● private void makeMove(int row, int col) {
78    if (playerXTurn) {
79        board[row][col] = Cell.X;
80    } else {
81        board[row][col] = Cell.O;
82    }
83}
84
85● private boolean isWon() {
86    for (int i = 0; i < 3; i++) {
87        if (board[i][0] != Cell.NotUsed && board[i][0] == board[i][1] && board[i][0] == board[i][2]) {
88            return true;
89        }
90        if (board[0][i] != Cell.NotUsed && board[0][i] == board[1][i] && board[0][i] == board[2][i]) {
91            return true;
92        }
93    }
94    if (board[0][0] != Cell.NotUsed && board[0][0] == board[1][1] && board[0][0] == board[2][2]) {
95        return true;
96    }
97    return board[0][2] != Cell.NotUsed && board[0][2] == board[1][1] && board[0][2] == board[2][0];
98}
99
100● private boolean isDraw() {
101    for (int i = 0; i < 3; i++) {
102        for (int j = 0; j < 3; j++) {
103            if (board[i][j] == Cell.NotUsed) {
104                return false;
105            }
106        }
107    }
108    return true;
109}
110
111● public static void main(String[] args) {
112}

```

```

105             return false;
106         }
107     }
108     return true;
109 }
110 }
111 public static void main(String[] args) {
112     Tictactoe game = new Tictactoe();
113     game.startGame();
114 }
115 }
116 }

```

Output:

The screenshot shows the Eclipse IDE interface with the Java editor and terminal output.

Java Editor (Top Left):

```

65 System.out.print(" 1 2 3 \n");
66 for (int i = 0; i < 3; i++) {
67     for (int j = 0; j < 3; j++) {
68         int row = scanner.nextInt();
69         int column = scanner.nextInt();
70         if (row == 0 && column == 0) {
71             board[i][j] = 'X';
72         } else if (row == 0 && column == 1) {
73             board[i][j] = 'O';
74         } else if (row == 0 && column == 2) {
75             board[i][j] = ' ';
76         } else if (row == 1 && column == 0) {
77             board[i][j] = ' ';
78         } else if (row == 1 && column == 1) {
79             board[i][j] = 'X';
80         } else if (row == 1 && column == 2) {
81             board[i][j] = 'O';
82         } else if (row == 2 && column == 0) {
83             board[i][j] = 'O';
84         } else if (row == 2 && column == 1) {
85             board[i][j] = 'X';
86         } else if (row == 2 && column == 2) {
87             board[i][j] = ' ';
88         }
89     }
90 }
91
92 // Check for win conditions
93 for (int i = 0; i < 3; i++) {
94     if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && board[i][0] != ' ') {
95         System.out.println("Player " + (board[i][0] == 'X' ? 2 : 1) + " wins!");
96         return;
97     }
98     if (board[0][i] == board[1][i] && board[1][i] == board[2][i] && board[0][i] != ' ') {
99         System.out.println("Player " + (board[0][i] == 'X' ? 2 : 1) + " wins!");
100        return;
101    }
102 }
103
104 // Check for draw
105 if (isFull()) {
106     System.out.println("It's a draw!");
107 }
108 }
109 }
110 }
111 public static void main(String[] args) {
112     Tictactoe game = new Tictactoe();
113     game.startGame();
114 }
115 }
116 }

```

Terminal Output (Bottom Right):

```

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

```