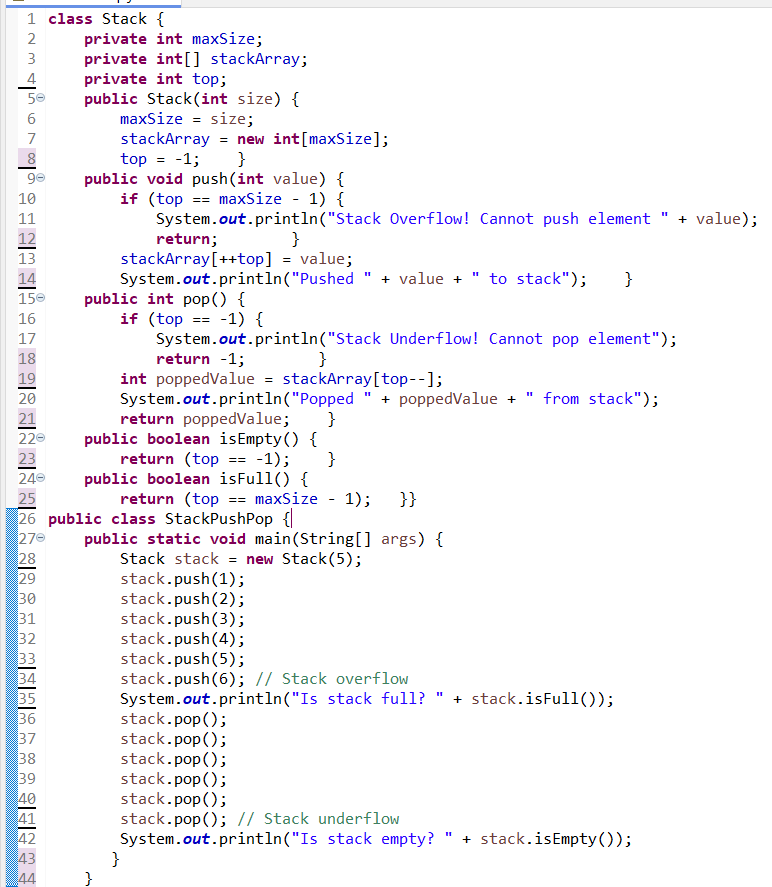
**PRACTICAL – 12**

**Aim:** - Write a Java program to implement Push and Pop operations of Stack. Also ensure stack overflow and underflow conditions are checked while performing push and pop operations respectively.

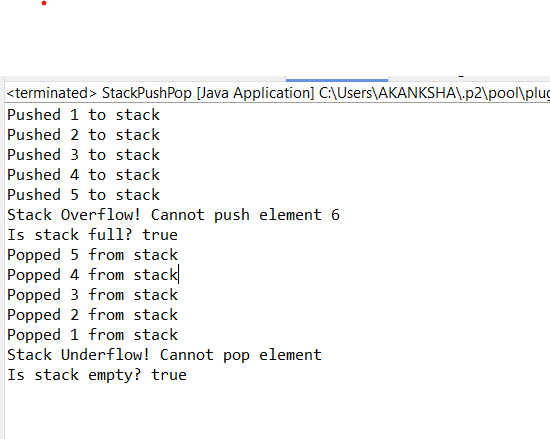
**Theory: -**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Code: -**

****

**Output: -**

****

**Learning Outcomes: -**

|  |
| --- |
|  |
|  |
|  |
|  |

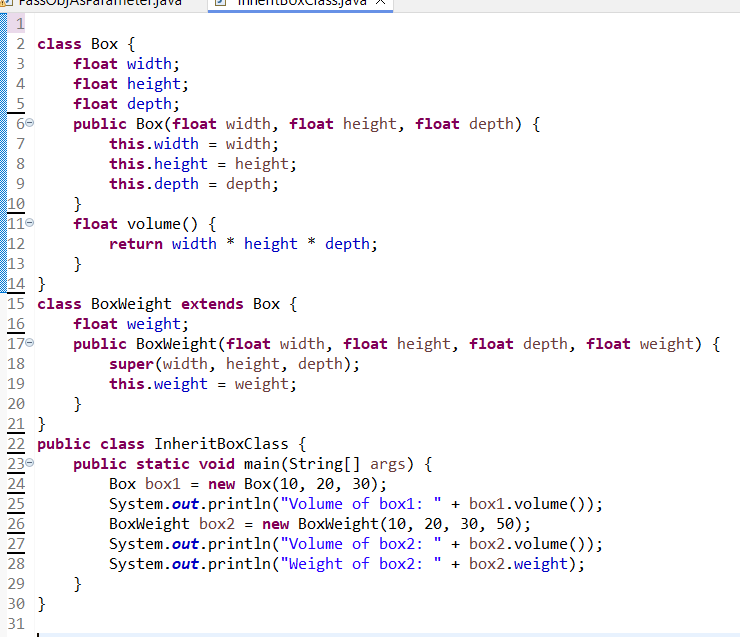
**PRACTICAL – 14**

**Aim: -** Write a Java program to implement inheritance Define a class box with the following instance variables: width, height and depth, all of type float. Create a new class Box Weight that extends Box to include weight as an instance variable. Write an application that tests the functionalities of both these classes.

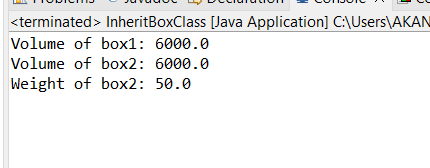
**Theory: -**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

**Code: -**

****

**Output: -**

****

**Learning Outcomes: -**

|  |
| --- |
|  |
|  |
|  |
|  |

**PRACTICAL – 15**

**Aim: -** Implement the following Java programs to demonstrate the concept of

exception handling using keywords try, catch, finally, throw and throws

wherever required.

**Theory: -**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**(A)**

(a) Write a Java program using switch to demonstrate the usage of

try/catch block for the following handling exceptions:

Case 1: Arithmetic Exception

Case 2. Index Out of Bounds Exception

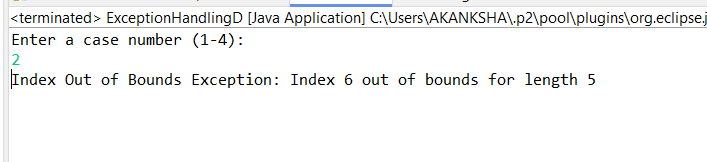
Case 3. Null Pointer Exception

Case 4. Number Format Exception

**Code: -**

****

**Output: -**

****

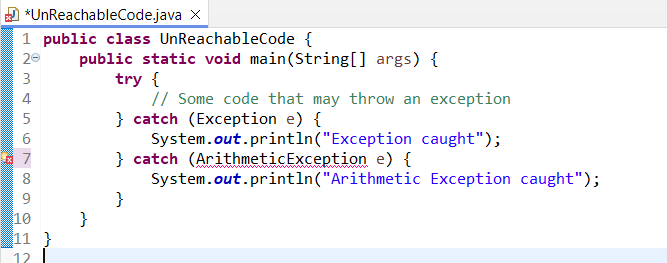
**(B)**

(b) Write a Java program demonstrate how unreachable code is created

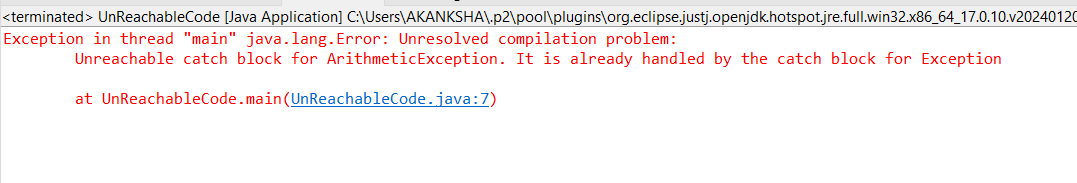
and compile-time error occurs when super class exception occurs prior

to subclass exception in series of catch statements.

**Code: -**

****

**Output: -**

****

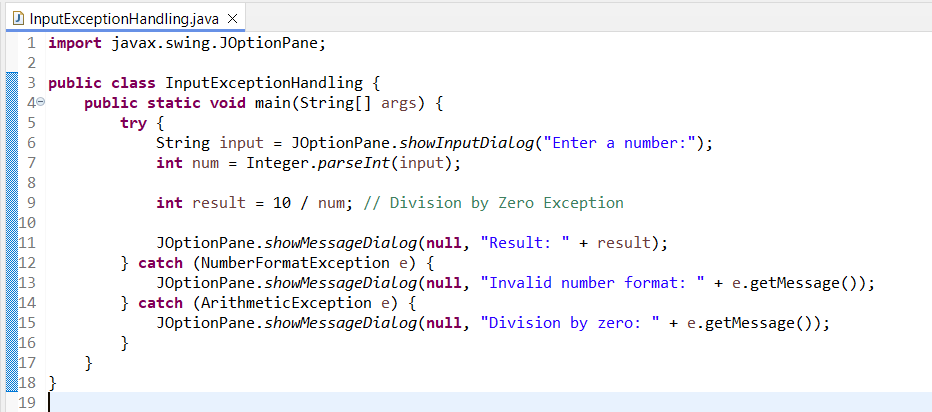
**(C)**

(c) Write a Java program that show to catch and handle number format

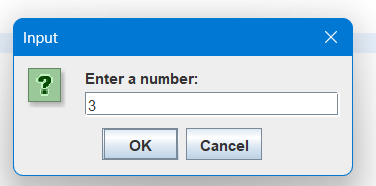
and divison by zero exceptions in programs that use input dialog boxes

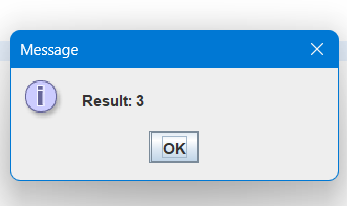
and/or text fields.

**Code: -**

****

**Output: -**

****

****

**Learning Outcomes: -**

|  |
| --- |
|  |
|  |
|  |
|  |

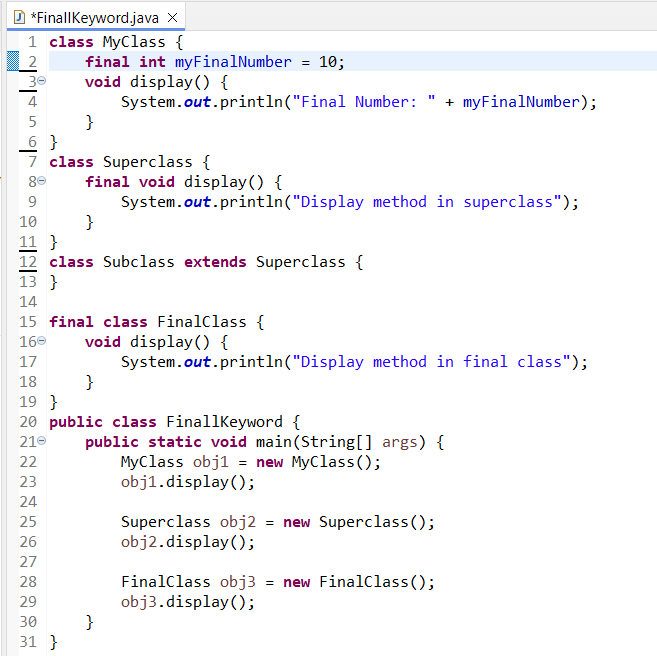
**PRACTICAL – 16**

**Aim:** - Demonstrate the use of final keyword with data member, function and class.

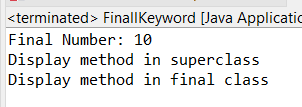
**Theory: -**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Code: -**

****

**Output: -**

****

**Learning Outcomes: -**

|  |
| --- |
|  |
|  |

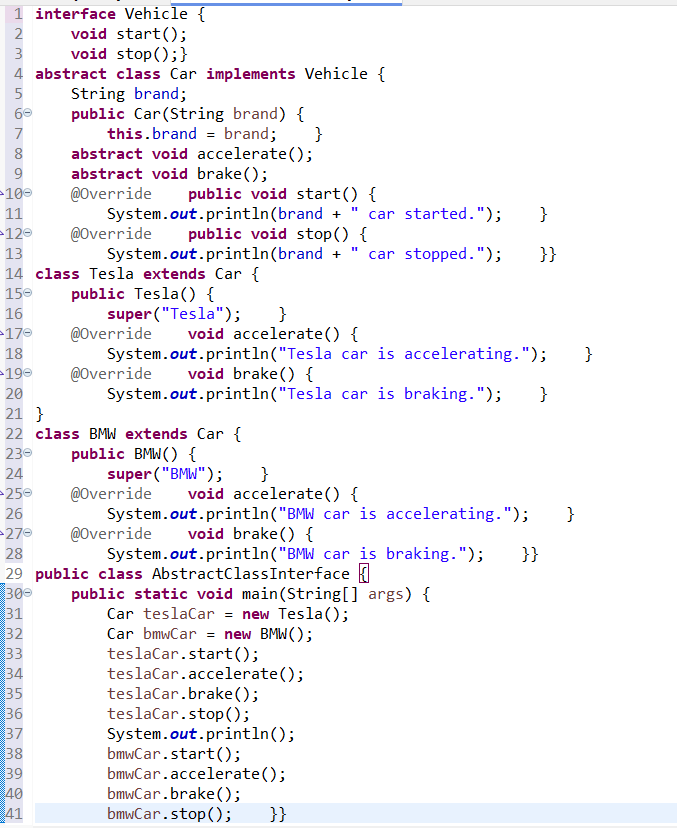
**PRACTICAL – 17**

**Aim: -** Write a Java program to demonstrate the concept of abstract classes and interfaces.

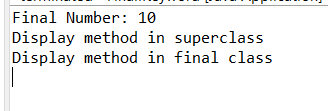
**Theory: -**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Code: -**

****

**Output: -**

****

**Learning Outcomes: -**

|  |
| --- |
|  |
|  |
|  |
|  |

**PRACTICAL – 18**

**Aim: -** Write a Java program to demonstrate the usage of following Collections:

List:

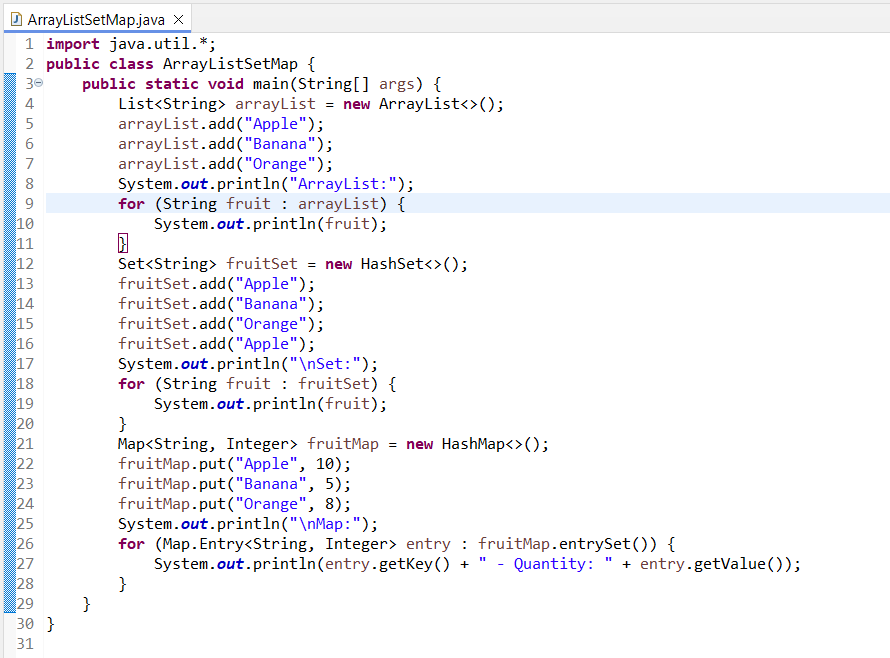
Array List

Set

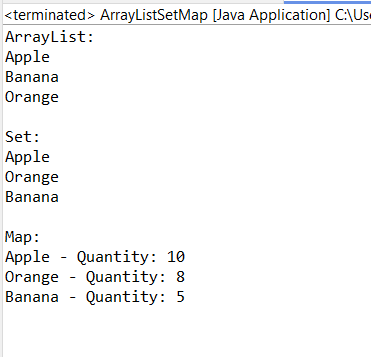
Map

**Theory: -**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

**Code: -**

**Output: -**

****

**Learning Outcomes: -**

|  |
| --- |
|  |
|  |
|  |
|  |

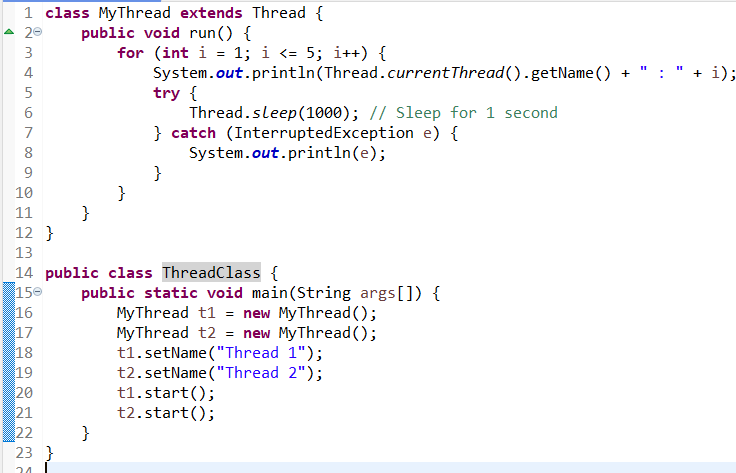
**PRACTICAL – 19**

**Aim: -** Design a program to demonstrate multi-threading using Thread Class.

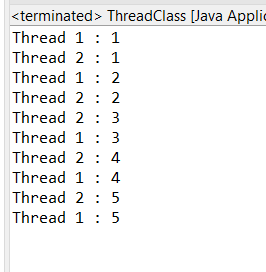
**Theory: -**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Code: -**

****

**Output: -**

****

**Learning Outcomes: -**

|  |
| --- |
|  |
|  |
|  |
|  |

**PRACTICAL – 20**

**Aim: -** Design a program to create game ‘Tic Tac Toe’.

**Theory: -**

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Code: -**

**import** java.util.Scanner;

**public** **class** TicTacToe {

**private** **char**[][] board;

**private** **char** currentPlayer;

**public** TicTacToe() {

board = **new** **char**[3][3];

currentPlayer = 'X';

initializeBoard();

}

// Initialize board with empty cells

**public** **void** initializeBoard() {

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

**for** (**int** j = 0; j < 3; j++) {

board[i][j] = '-';

}

}

}

// Print the current board

**public** **void** printBoard() {

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

**for** (**int** j = 0; j < 3; j++) {

System.***out***.print(board[i][j] + " ");

}

System.***out***.println();

}

}

// Switch player turns

**public** **void** switchPlayer() {

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

}

// Place a move on the board

**public** **boolean** placeMove(**int** row, **int** col) {

**if** (row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] == '-') {

board[row][col] = currentPlayer;

**return** **true**;

} **else** {

**return** **false**;

}

}

// Check if the game is over

**public** **boolean** isGameOver() {

**return** (checkRows() || checkColumns() || checkDiagonals() || isBoardFull());

}

// Check rows for win

**private** **boolean** checkRows() {

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

**if** (board[i][0] != '-' && board[i][0] == board[i][1] && board[i][0] == board[i][2]) {

**return** **true**;

}

}

**return** **false**;

}

// Check columns for win

**private** **boolean** checkColumns() {

**for** (**int** j = 0; j < 3; j++) {

**if** (board[0][j] != '-' && board[0][j] == board[1][j] && board[0][j] == board[2][j]) {

**return** **true**;

}

}

**return** **false**;

}

// Check diagonals for win

**private** **boolean** checkDiagonals() {

**return** (board[0][0] != '-' && board[0][0] == board[1][1] && board[0][0] == board[2][2]) ||

(board[0][2] != '-' && board[0][2] == board[1][1] && board[0][2] == board[2][0]);

}

// Check if the board is full

**private** **boolean** isBoardFull() {

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

**for** (**int** j = 0; j < 3; j++) {

**if** (board[i][j] == '-') {

**return** **false**;

}

}

}

**return** **true**;

}

// Main method to run the game

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

Scanner scanner = **new** Scanner(System.***in***);

TicTacToe game = **new** TicTacToe();

**boolean** gameOver = **false**;

**while** (!gameOver) {

System.***out***.println("Current board:");

game.printBoard();

System.***out***.println("Player " + game.currentPlayer + ", enter your move (row[0-2] column[0-2]):");

**int** row = scanner.nextInt();

**int** col = scanner.nextInt();

**if** (game.placeMove(row, col)) {

**if** (game.isGameOver()) {

gameOver = **true**;

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

} **else** **if** (game.isBoardFull()) {

gameOver = **true**;

System.***out***.println("It's a draw!");

} **else** {

game.switchPlayer();

}

} **else** {

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

}

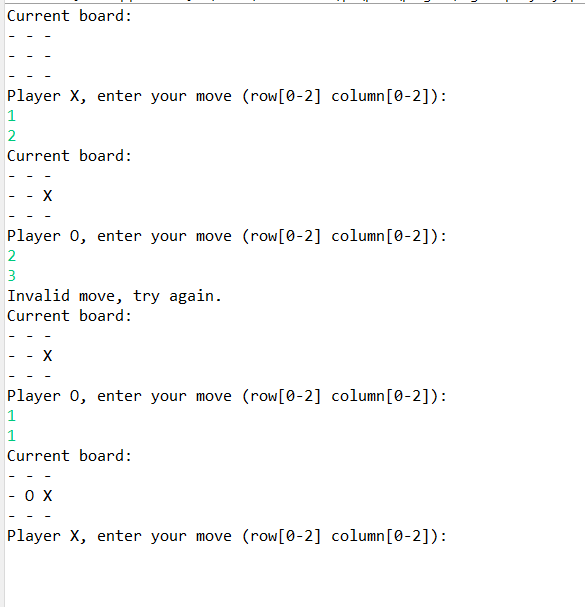
}

scanner.close();

}

}

**Output: -**

****

**Learning Outcomes: -**

|  |
| --- |
|  |
|  |
|  |
|  |