The following documentation of the proposed classes and methods are in this format:

**Package “” {** The package that the following classes are nested in

**ClassName**: What the class is for (purpose and responsibilities)

**Data Members**:

Visibility, attributes, name - purpose

**Methods**

Visibility, attributes, return type, name(with parameters)

**Description**:

**Parameters with definitions:**

**Returns:** Type and representation

**Exceptions thrown**: if applicable

**}**

**Package “app” {**

**Driver:** Starts the application

**Data Members:**

**None**

**Methods:**

**private static void main(String[] args)**

**Description:** Makes a static call to NimController.runApp() to initiate the game.

**Parameters:** Specified starting parameters. (unused in this application)

**Returns:** Void

**NimController:** Controls the flow and operation of the application

**Data Members:**

**private static turnVal -** The integer value of the current player’s array index using the TURN\_VAL final integers

**private static difficulty** – The Difficulty Enum value for the current game

**private static players** – The array of current players in the game

**private static gameBoard** – The current game board instance being played

**private static instructionBox** – The custom InstructionBox object to display gameplay instructions and rules to the user

**private static PLAYER\_ONE\_TURN\_VAL** – The un-changeable integer value of player ONE’S index in the players array

**private static PLAYER\_ONE\_TURN\_VAL** – The un-changeable integer value of player TWO’S index in the players array

**Methods:**

**public static void runApp()**

**Description:** The primary method for controlling the operation of the application

**Parameters:** None

**Returns:** Void

**Private static void playGameTurn()**

**Description:** Controls the logic for one single turn during game play.

Controls interaction logic for either human or computer player and the gameBoard

**Parameters:** None

**Returns:** Void

**private static boolean isValidMoveInput(String userInput)**

**Description:** Determines if the String retrieved from the user can be parsed into a valid game move. Input should be in a comma-separated format. (ie: “2, 4”)

**Note:** Spacing shall not affect proper parsing of the string

**Parameters:**

**userInput** - The user’s String to be checked

**Returns:**

True**,** if the user’s String IS valid

False**,** if the user’s String IS NOT valid

**private static isValidMove(int rowNumber, int tokenCount)**

**Description:** Determines if the desired move on the gameBoard is possible. Checks that the rowNumber exists and that the tokenCount is within the acceptable range of that rowNumber. Which is 1-(n), where (n) is the number of tokens currently in that rowNumber

**Parameters:**

**rowNumber** – The integer value of a selected row

**tokenCount** – The integer value representing the number of tokens selected to be removed from a selected row

**Returns:**

True, if the move IS possible

False, if the move IS NOT possible

**private static int[] generateValidRandomMove()**

**Description:** Randomly generates a move which will be both valid and possible during the current game state

**Note:** “Valid” means the row value exists in the current game, and “Possible” means the token value is between 1 and the amount of tokens in the row

**Parameters:** None

**Returns:** Integer array of size 2 in the format of: [rowNumber,tokenCount]

**private static boolean isGameOver()**

**Description:** Checks if the current player took the last token

**Parameters:** None

**Returns:** True, if the current player lost

False, if the game has not been lost.

**private static boolean isValidDifficultyChoice(String userInput)**

**Description:** Determines if the user’s input is able to be parsed into a

Difficulty Enum value

**Parameters:**

**userInput** - The user’s String input to be checked

**Returns:**

True, if a valid Difficulty CAN be parsed from the input text

False, if a valid Difficulty CAN NOT be parsed from the input text

**private static boolean isValidPlayerName(String userInput)**

**Description:** Determines if the user’s input is not null nor empty

**Parameters:**

**userInput** - The user’s String input to be checked

**Returns:**

True, if the user’s input IS NOT null nor empty

False, if the user’s input IS null nor empty

**private static boolean isValidInt(String userInput)**

**Description:** Determines if the user’s input can be parsed into an

integer value

**Parameters:**

**userInput** - The user’s String input to be checked

**Returns:**

True, if the user’s input CAN be parsed

False, if the user’s input CAN NOT be parsed

**}**

**Package “view” {**

**Console**: Controls the way the application and user’s interact in the windows console

**Data Members**: None

**Methods**:

**public static void writeLine(String message)**

**Description:** Writes a single message to the console with a

carriage return

**Parameters:**

**message** – The string message to be displayed to the user

**Returns:** Void

**public static void writecollection(ArrayList<T> messages)**

**Description:** Writes each message from a collection. Each message is

written on its own line and is followed by a carriage

return

**Parameters:**

**messages** – The collection of messages to be displayed to the

user

**Returns:** Void

**public static String readLine()**

**Description:** Reads a single line as String input from the windows

Console

**Parameters:** None

**Returns:** The String retrieved from the windows console

representing the user’s choice

**public static String promptUserForInput(String prompt)**

**Description:** Asks the user a question and waits for their response

**Parameters:** The String type question to pose to the user via the

windows console as a string

**Returns:**  The String retrieved from the windows console

representing the user’s choice

**public static String promptUserForMenuChoice(String[] options)**

**Description:** Prints out a collection of options for the user to choose

from, and waits for their response

**Parameters:** String[] containing the collections of options to display

to the user at the windows console

**Returns:** The String retrieved from the windows console

representing the user’s choice

**InstructionBox:** A simple stage to display information to the user throughout the application runtime

**Data Members:**  None

**Methods:**

**public static void show(String displayData)**

**Description:**  Displays an un-closeable window to display the given String value

**Parameters:** **displayData –** The String text to display to the user

**Returns:** Void

**public static void show(ArrayList<String> displayData)**

**Description:**  Displays an un-closeable window to display the given collection of Strings to the user

**Parameters:** **displayData –** The collection of Strings to display to the user

**Returns:** Void

**}**

**Package “models” {**

**GameBoard:** Represents the object which is the rows, and tokens in the rows, for game play

**Data Members:** None

**Methods:**

**public GameBoard(Difficulty difficulty)**

**Description:** Constructor to create the game board object with which the game play will interact

**Parameters:** **difficulty –** The Difficulty Enum type used to create the game board

**Returns:** Constructor, implicitly returns the new object

**public int getRowTokenValue(int rowNumber)**

**Description:** Gets the amount of tokens in a given row number

**Parameters:** The integer value of the row to count

**Returns:** The integer value representing the number of tokens in

that row

**public int getNumOfRows()**

**Description:** Gets the amount of rows which make up the game board

**Parameters:** None

**Returns:** The integer value of the number of rows on the game board

**public void takeTokens(int rowNumber, int numOfTokens)**

**Description:** Removes the number of tokens specified by numOfTokens from the given row specified by rowNumber

**Parameters:** **rowNumber –** The integer value of the row to access

**numOfTokens –** The integer value of the amount of tokens to remove from the row

**Returns:** Void

**@Override public String toString()**

**Description:** Creates a string representation of the current game board in a format which is displayable to the UI

**Parameters:** None

**Returns:** The resulting string representation of the current state of the board

**Player:** Represents an actual Player of the game, whether computer or human player

**Data Members: playerName –** A String value for the name of the Player

**isHumanPlayer** – A boolean value for the player type. True for human players, false for computer players

**Methods:**

**public Player(String name)**

**Description:**  Constructor to stand up a new Person object

**Parameters: name –** A string representing the player’s name

**Returns:** Constructor, implicitly returns the new object

**public String getName()**

**Description:** Gets the current name of the player

**Parameters:** None

**Returns:** The String representing the player’s name

**public void setName(String name)**

**Description:** Sets the current name of the player

**Parameters: name –** A string to store as the player’s name

**Returns:** Void

**public void setIsHumanPlayer(boolean isHumanPlayer)**

**Description:** Sets the current human status of the player

**Parameters:**  **isHumanPlayer** – boolean representing if the player is a

human or computer

**Returns:** Void

**@Override public String toString()**

**Description:** Creates a string to displaying player attributes

**Parameters:** None

**Returns:** The created String as representation of the player

Object

**}**

**Package “enums” {**

**Difficulty:** Enumeration of the set difficulty parameters as specified by the client documents

**Data Members:**

**private int[] rowTokenCounts –** The amounts of tokens in each row

**EASY(int, int) –** The easy difficulty setting which has two rows of tokens

**MEDIUM(int, int, int) –** The medium difficulty setting which has three rows of tokens

**HARD(int, int, int, int) –** The hard difficulty setting which has four rows of tokens

**Methods:**

**private Difficulty(int[] rowTokenCounts)**

**Description:** Constructor to create the attributed enum type. Assigns the integer array of token counts to the difficulty enum

**Parameters:** **rowTokenCounts –** The int[] representing the number of tokens for each row, with each index of the int[] being a different row

**Returns:** Constructor, implicitly returns the new enum object

**}**