Programming Assignment 1

Graded

Student

Mohammed Ali Abdul-nabi

Total Points

19 / 20 pts

Autograder Score

10.0 / 10.0

Passed Tests

Code compiled Succesfully

HashDictionary Tests

Configurations Tests

Question 2

HashDictionary **3** / 4 pts

2.1 **All methods of Data.class implemented correctly** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Wrong

2.2 **All methods of HashDictionary and primarily hashFunction. HashDictionary does not use hashCode and is an Array of List implemented correctly** **2.5** / 3.5 pts

– 0 pts Correct

– 2 pts Wrong hash function implementation.

✔ **– 1 pt** Partially Correct

– 3.5 pts No submission

As taught in class, it is always better to use a large prime number as hash multiplier.

Question 3

Configurations **4** / 4 pts

3.1 **Method Wins is implemented correctly and takes into account all possible scenarios(Column, Row, Diagonal and inverse Diagonal** **3** / 3 pts

✔ **– 0 pts** Correct

– 1 pt missing one scenario

– 2 pts missing two scenarios

– 3 pts missing implementation

3.2 **All methods are implemented as required** **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt No submission

– 0.5 pts Partially Correct

Question 4

Programming Style Points **2** / 2 pts

4.1 **Meaningful names for variables and constants: All instance variables are private and all are required** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts methods missing/public instance variables/unnecessary instance variables

4.2 **Readability: Good Indentation** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts No identation

4.3 **Code Comments : Instance variables and methods, Comments within code to explain algorithms** **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Missing comments.

– 0.5 pts Partially Correct

+ 5 pts This is only for specific cases where the autograder was not able to the grade the submission but the code was working on student's local machine.

Autograder Results

Autograder Output

Note: HashDictionary.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

Code compiled Succesfully

HashDictionary Tests

Configurations Tests

Submitted Files

Data.java Download

```java
public class Data {
    private String config;
    private int score;

    // Constructor to initialize Data objects
    public Data(String config, int score) {
        this.config = config;
        this.score = score;
    }

    // Getter method for configuration, returns the configuration stored in Data object.
    public String getConfiguration() {
        return config;
    }

    // Getter method for score, returns the score in this Data.
    public int getScore() {
        return score;
    }
}

```

HashDictionary.java

Download

```java
import java.util.LinkedList;

public class HashDictionary implements DictionaryADT {
    // Array LinkedLists to implements chaining for hash collisions
    private LinkedList<Data>[] table;
    private int size;

    //Constructor initalizes hash table with specified size
    public HashDictionary(int size) {
        this.size = size;
        table = (LinkedList<Data>[]) new LinkedList[size];  // Safe cast

        for (int i = 0; i < size; i++) {
            table[i] = new LinkedList<>();
        }
    }

    //Computes the index of a key
    private int hashFunction(String key) {
        int hash = 0;
        int prime = 10;  // Polynomial hash with a small prime number
        //Generate a hashvalue
        for (int i = 0; i < key.length(); i++) {
            hash = (prime * hash + key.charAt(i)) % size;
        }
        return hash;
    }

    //Adds record to the dictionary
    @Override
    public int put(Data record) throws DictionaryException {
        int index = hashFunction(record.getConfiguration());

        for (int i = 0; i < table[index].size(); i++) {
            // Get element at index i
            Data data = table[index].get(i);
            if (data.getConfiguration().equals(record.getConfiguration())) {
                // Throw exception if configuration exists in dictionary
                throw new DictionaryException();
            }
        }
        table[index].add(record);  // Add the new record to the chain
        return table[index].size() > 1 ? 1 : 0;  // Return 1 if there was a collision, 0 otherwise
    }



    @Override
    //Removes record from hashtble
    public void remove(String config) throws DictionaryException {
```

```
50        int index = hashFunction(config);
51
52        //Iterate throught the chain
53        for (int i = 0; i < table[index].size(); i++) {
54            Data data = table[index].get(i);  // Get each element by index
55            if (data.getConfiguration().equals(config)) {   //Compare and each if we found it
56                table[index].remove(i);  // Remove element at the current index
57                return;  // Exit after removing the element
58            }
59        }
60        throw new DictionaryException();  // Throw exception if config not found
61    }
62
63
64    @Override
65    //Returns the score given config
66    public int get(String config) {
67        int index = hashFunction(config);
68
69        // Iterate thrugh the chain
70        for (int i = 0; i < table[index].size(); i++) {
71            Data data = table[index].get(i);  // Get each element by index
72            if (data.getConfiguration().equals(config)) {
73                return data.getScore();  // Return the score if found
74            }
75        }
76        return -1;  // Configuration not found
77    }
78
79
80    @Override
81    //Number of objects in the dictionary
82    public int numRecords() {
83        int numRecords =0;  //intialize numRecords
84        //Iterate throught the table
85        for (int i = 0; i < table.length; i++){
86            //Add the size when not null
87            if (table[i] != null){
88                numRecords += table[i].size();
89            }
90        }
91
92        return numRecords;//Return the total size.
93    }
94}
95
```

Configurations.java Download

```java
public class Configurations {
    private char[][] board;    //2D array to show board
    private int size;          //Size of board
    private int lengthToWin;   //k needed to win
    private int maxLevels;     //Maxium levels of game tree

    //Constrructor that intialized the game board and the game settings
    public Configurations(int boardSize, int lengthToWin, int maxLevels) {
        //Initalize the variables
        this.size = boardSize;
        this.lengthToWin = lengthToWin;
        this.maxLevels = maxLevels;
        board = new char[size][size];
        //Make all positions as empty
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                board[i][j] = ' ';
            }
        }
    }

    //Create hash dictionary of size
    public HashDictionary createDictionary() {
        return new HashDictionary(6001);  // A prime number size
    }

    //Checks if current configuation of the board is in the table
    public int repeatedConfiguration(HashDictionary hashTable) {
        String config = boardToString();
        //Returns score if found int the table
        return hashTable.get(config);
    }

    //Add current config and score to the table
    public void addConfiguration(HashDictionary hashTable, int score) {
        String config = boardToString();
        try {
            hashTable.put(new Data(config, score));
        } catch (DictionaryException e) {
            System.out.println(e.getMessage());
        }
    }

    //Places the X or O on the game board at specified loaction
    public void savePlay(int row, int col, char symbol) {
        board[row][col] = symbol;
    }

    //Checks the selected square is empty.
```

```
50      public boolean squareIsEmpty(int row, int col) {
51          return board[row][col] == ' ';
52      }
53
54      // Check all possible diagonals for a win
55      private boolean checkAllDiagonals(char symbol) {
56          // Check all down-right diagonals (from top row and left column)
57          for (int i = 0; i < size; i++) {
58              if (checkLine(symbol, i, 0, 1, 1) ||  // Diagonals stating from left col
59                  checkLine(symbol, 0, i, 1, 1)) {  // Diagonals starting from top row
60                  return true;                                  // Win detected
61              }
62          }
63
64          // Check all down-left diagonals (from top row and right column)
65          for (int i = 0; i < size; i++) {
66              if (checkLine(symbol, i, size - 1, 1, -1) ||    // Diagonals stating from right col
67                  checkLine(symbol, 0, i, 1, -1)) {      // Diagonals starting from top row
68                  return true;                                   // Win detected
69              }
70          }
71          return false;   //Win not found on the diags
72      }
73
74      // Checks if the given symbol (X or O) has formed a winning line
75      public boolean wins(char symbol) {
76          // Check all rows and columns for a win
77          for (int i = 0; i < size; i++) {
78              if (checkLine(symbol, i, 0, 0, 1) ||  // Check row i
79                  checkLine(symbol, 0, i, 1, 0)) {  // Check column i
80                  return true;
81              }
82          }
83
84          // Check all possible diagonals
85          return checkAllDiagonals(symbol);
86      }
87
88      // Helper method to check a line (row, column, or diagonal) for a win
89      private boolean checkLine(char symbol, int row, int col, int deltaRow, int deltaCol) {
90          int count = 0;// Count of consecutive symbol
91          while (row >= 0 && row < size && col >= 0 && col < size) {
92              if (board[row][col] == symbol) {
93                  //If symbol matches we increment count
94                  count++;
95                  if (count >= lengthToWin) {
96                      //Win foiund
97                      return true;
98                  }
99              } else {
100                 // Reset count if does not match
101                 count = 0;
```

```
            }
            row += deltaRow;
            col += deltaCol;
        }
        // No win detected in this line
        return false;
    }

    //Checks game ends in draw
    public boolean isDraw() {
        //Iterate and check over empty spaces
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                //The board is not empty, not draw
                if (board[i][j] == ' ') {
                    return false;
                }
            }
        }
        //Game ended in a draw
        return true;
    }

    //Checks the game board and will return the corresponding score
    public int evalBoard() {
        if (wins('O')) return 3;  // Computer wins
        if (wins('X')) return 0;  // Human wins
        if (isDraw()) return 2;        // Draw
        return 1;                  // Undecided
    }

    //Convert board into a string
    private String boardToString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                sb.append(board[i][j]);
            }
        }
        return sb.toString();
    }
}
```