Name:Manikandan Thyagesh **Tutorial Group ID**: T07	{Insert a recent photo of you here Use real and recent photos that shows the face clearly. CS2103/T is a big class; seeing your photo here helps us connect the face to the name better. }

Code

```
/* @author Thyagesh Manikandan (A0100124J) */
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.Scanner;
import java.util.LinkedList;
import java.util.concurrent.LinkedBlockingQueue;

import static java.lang.System.out;

/**
 *  This class runs the whole TextBuddy program which is based on a few
 *  simple commands: - add, delete, clear, display and exit. The
 *  functionality of the program is based on this sample input/output
 *  given:
 *
 *       c:> TextBuddy mytextfile.txt (OR c:>java TextBuddy mytextfile.txt)
 *       Welcome to TextBuddy. mytextfile.txt is ready for use
 *       command: add little brown fox
 *       added to mytextfile.txt: “little brown fox”
 *       command: display
 *       1. little brown fox
 *       command: add jumped over the moon
 *       added to mytextfile.txt: “jumped over the moon”
 *       command: display
 *       1. little brown fox
 *       2. jumped over the moon
 *       command: delete 2
 *       deleted from mytextfile.txt: “jumped over the moon”
 *       command: display
 *       1. little brown fox
 *       command: clear
 *       all content deleted from mytextfile.txt
 *       command: display
 *       mytextfile.txt is empty
 *       command: exit
 *       c:>
 *
 *  ********************************************************************
 *                       ********Assumptions*********
 *  - if an unknown command is given, let the user know that it is an unknown command
 *  - if the given file has problems or the program is unable to create a file
 *  with the given name, exit gracefully
 *  - if "display" is followed by any other characters, just display
 *  - user command is not case sensitive
 *  - an add command with no parameters is taken as a newline request
 *  - if the given files exists, load the data from it so that an
 *  initial display call can be made
 */
public class TextBuddy implements Runnable
{

   /* Class Attributes
    * a string for the name file being used
    * a BufferedWriter object to allow for writing to/saving the file
    * a linked-list of strings for quick local executions before saving into file
    * a Scanner object to parse the user input
    * an autosaver thread that does the actual modification of the file
    * a linkedblockingqueue to save the changes which is then read by our autosaver while saving
    * _hasNotClosed is so that housekeeping is not re-done
    * _hasEnded to notify the autosaver that the program has been terminated
    */
   private String _fileName;
   private BufferedWriter _bufFileWriter = null;
   private LinkedList<String> _localCopyOfFileData = new LinkedList<String>();
   private Scanner _inputStream = new Scanner(System.in);
```

```
private Thread _autoSaver = null;
private LinkedBlockingQueue<UserCommandInterpretter> _unsavedChanges = new LinkedBlockingQueue<UserCommandInterpretter>();
private boolean _hasNotClosed = true;
private boolean _hasEnded = false;

/*
 * String attributes for general messages to user
 */
private static final String MESSAGE_INTERNAL_ERROR_TEMPLATE = "An iternal error has occurred%s! We apologise for the inconvenience caused!";
private static final String MESSAGE_INTERNAL_ERROR = String.format(MESSAGE_INTERNAL_ERROR_TEMPLATE, "");
private static final String MESSAGE_INTERNAL_ERROR_SAVE = String.format(MESSAGE_INTERNAL_ERROR_TEMPLATE, " while trying to save the changes to file");

private static final String MESSAGE_REQUEST_USER_FOR_COMMAND = "command: ";
private static final String MESSAGE_GREET_USER_TEMPLATE = "Welcome to TextBuddy. %s is ready for use";
private final String MESSAGE_GREET_USER;

private static final String MESSAGE_FEEDBACK_FOR_ADD = "added to %s: \"%s\"";
private static final String MESSAGE_FEEDBACK_FOR_DELETE = "deleted from %s: \"%s\"";
private static final String MESSAGE_FEEDBACK_FOR_CLEAR = "all content deleted from %s";
private static final String MESSAGE_FEEDBACK_FOR_EMPTY_DISPLAY = "%s is empty";

/* Class Methods */

public TextBuddy (String fileName) throws IOException, FileNotFoundException, SecurityException
{
   validateAndSaveParameter(fileName);
   this.initialiseFileHandling(fileName);
   this.MESSAGE_GREET_USER = String.format(MESSAGE_GREET_USER_TEMPLATE, this._fileName);
   this.initialiseAutoSaver();
}

private void validateAndSaveParameter (String fileName)
{
   boolean isValidParameter = fileName.length() > 0;
   if (isValidParameter)
   {
      this._fileName = fileName;
      return;
   }
   this.informUser("Given paramter does not contain a file name!");
   System.exit(0);
}

/*
 * Handles all the file related initializations like the attribute
 * _bufFileWriter
 */
public void initialiseFileHandling (String fileName) throws IOException, SecurityException, FileNotFoundException
{

   File givenFile = new File(this._fileName);
   this.readExistingDataFromFile(givenFile);
   /* Initialize the buffer Writer for saving */
   this._bufFileWriter = new BufferedWriter(new FileWriter(givenFile, true));
   return;

}

/* Starts a new thread that does the auto-saving */
private void initialiseAutoSaver ()
{
   this._autoSaver = new Thread(this);
   _autoSaver.setPriority(Thread.MAX_PRIORITY);
}

/*
 * Loads the existing data from the file and fills the local copy with the
 * data
 */
private void readExistingDataFromFile (File givenFile) throws IOException, SecurityException, FileNotFoundException
{

   if (givenFile.exists())
   {
      BufferedReader bufFileReader = new BufferedReader(new FileReader(givenFile));

      String lineReader = "";
      while ((lineReader = bufFileReader.readLine()) != null)
      {
         this._localCopyOfFileData.add(lineReader);
      }
      bufFileReader.close();
   }
   else if (givenFile.createNewFile())
   {
      /* Everything's fine, we can return normally */
      return;
   }
   else
   {
      throw new IOException("Could not create a new file in the disk! Permissions not given!");
   }
}

/* A simple function to kick start TextBuddy */
public void start ()
{
   _autoSaver.start();
   this.interactWithUserUntilExitCommand();
   this.close();
}

/*
 * Greets the user and calls the appropriate functions to be executed
 */
public void interactWithUserUntilExitCommand ()
```

```
{
   this.informUser(MESSAGE_GREET_USER);

   while (true)
   {
      UserCommandInterpretter newCommand = getNewUserCommand();

      switch (newCommand.getCommand())
      {
         case ADD:
            this.handleAddCommand(newCommand);
            break;
         case DELETE:
            this.handleDeleteCommand(newCommand);
            break;
         case DISPLAY:
            this.handleDisplayCommand();
            break;
         case CLEAR:
            this.handleClearCommand(newCommand);
            break;
         case UNKNOWN:
            this.handleUnknownCommand(newCommand);
            break;
         case EXIT:
            this.handleExitCommand(newCommand);
            return;
         default:
            out.printf(MESSAGE_INTERNAL_ERROR_TEMPLATE, " while trying to comprehend user's command");
            System.exit(0);
      }
   }
}

private UserCommandInterpretter getNewUserCommand ()
{
   this.informUser(MESSAGE_REQUEST_USER_FOR_COMMAND);
   String command = _inputStream.nextLine();

   UserCommandInterpretter newCommand = new UserCommandInterpretter(command);
   return newCommand;
}

/* Handler functions */
private void handleAddCommand (UserCommandInterpretter addCommand)
{
   String toAdd = addCommand.getStringParameter();

   this._localCopyOfFileData.add(toAdd);
   this.logTheChange(addCommand);

   String feedbackMessage = String.format(MESSAGE_FEEDBACK_FOR_ADD, this._fileName, toAdd);
   this.informUser(feedbackMessage);
}

public void handleDeleteCommand (UserCommandInterpretter deleteCommand)
{
   String feedbackMessage = "";
   int indexToDelete = deleteCommand.getIntParameter();
   int lengthOfAvailableData = this._localCopyOfFileData.size();

   if (indexToDelete == deleteCommand.WRONG_VALUE_GIVEN)
   {
      feedbackMessage = deleteCommand.getErrorString();

   }
   else if (indexToDelete < 1 || indexToDelete > lengthOfAvailableData)
   {
      feedbackMessage = "Given index is out-of-range of the available number of lines!";
   }
   else
   {
      this.logTheChange(deleteCommand);
      String deletedString = _localCopyOfFileData.remove(indexToDelete - 1);

      feedbackMessage = String.format(MESSAGE_FEEDBACK_FOR_DELETE, this._fileName, deletedString);
   }
   informUser(feedbackMessage);
}

public void handleDisplayCommand ()
{
   String feedbackMessage = "";
   boolean isFileEmpty = this._localCopyOfFileData.size() == 0;

   if (isFileEmpty)
   {
      feedbackMessage = String.format(MESSAGE_FEEDBACK_FOR_EMPTY_DISPLAY, this._fileName);
   }
   else
   {
      for (int index = 0 ; index < _localCopyOfFileData.size() ; index++)
      {
         feedbackMessage += (index + 1) + ". " + _localCopyOfFileData.get(index) + System.lineSeparator();
      }
   }
   /* trim() to remove the trailing new line */
   informUser(feedbackMessage.trim());
}

public void handleClearCommand (UserCommandInterpretter clearCommand)
{
   String feedbackMessage = String.format(MESSAGE_FEEDBACK_FOR_CLEAR, this._fileName);

   this._localCopyOfFileData.clear();
   this.logTheChange(clearCommand);
```

```
   informUser(feedbackMessage);
}

private void handleExitCommand (UserCommandInterpretter exitCommand)
{
   this.logTheChange(exitCommand);
}

private void handleUnknownCommand (UserCommandInterpretter unknownCommand)
{
   String errorString = unknownCommand.getErrorString();
   this.informUser(errorString);
}

/* Takes a message and informs the user appropriately */
public void informUser (String message)
{
   out.println(message);
}

/* Logs the new command for it to be saved */
private void logTheChange (UserCommandInterpretter newCommand)
{
   try
   {
      this._unsavedChanges.put(newCommand);
   }
   catch (InterruptedException ie)
   {
      this.informUser("Interrupted while logging the change!");
      return;
   }
}

/* The Auto-save thread's run method */
public void run ()
{
   this.saveToFile();
}

/* Runs infinitely saving any changes made until program ends */
public void saveToFile ()
{
   try
   {
      UserCommandInterpretter logData;
      while (true)
      {
         logData = this._unsavedChanges.take();

         /*
          * a performance enhancement where regardless of how many items
          * are left in the queue, if the program is over, write the
          * local copy to the file
          */
         if (logData._command != COMMAND_TYPE.EXIT && this._hasEnded)
         {
            this.writeLocalCopyToFile();
            return;
         }
         else
         {
            switch (logData._command)
            {
               case ADD:
                  this._bufFileWriter.write(logData._strparam);
                  this._bufFileWriter.newLine();
                  break;
               case DELETE:
                  int lineNumber = logData._intparam;
                  this.deleteFromFile(lineNumber);
                  break;
               case CLEAR:
                  this._bufFileWriter.close();
                  this._bufFileWriter = new BufferedWriter(new FileWriter(new File(this._fileName)));
                  break;
               case EXIT:
                  return;
               default:
                  out.printf(MESSAGE_INTERNAL_ERROR_SAVE);
                  System.exit(0);
            }
            _bufFileWriter.flush();
         }// else condition
      }// while loop
   }// try clause
   catch (IOException io)
   {
      this.informUser(this._fileName + " can no longer be modified by the program! Please restart the program!");
   }
   catch (InterruptedException ie)
   {
      out.printf(MESSAGE_INTERNAL_ERROR_SAVE);
   }
   finally
   {
      try
      {
         _bufFileWriter.flush();
      }
      catch (IOException io)
      {
         out.printf(MESSAGE_INTERNAL_ERROR_SAVE);
      }
   }
```

```
    return;
}

/* Deletes a particular line given the number from the file */
private boolean deleteFromFile (int lineNumber)
{

    File tempFile = null;
    File oldFile = new File(this._fileName);
    BufferedReader bufTempReader = null;
    BufferedWriter bufTempWriter = null;
    String tempLineReader;

    try
    {
        tempFile = File.createTempFile("temp_TextBuddy", "");
        bufTempWriter = new BufferedWriter(new FileWriter(tempFile));
        bufTempReader = new BufferedReader(new FileReader(oldFile));

        // copy all the lines up to the line to delete to a temporary file
        for (int i = 1 ; i < lineNumber ; i++)
        {
            if ((tempLineReader = bufTempReader.readLine()) != null)
            {
                bufTempWriter.write(tempLineReader);
                bufTempWriter.newLine();
            }
            else
            {
                out.printf(MESSAGE_INTERNAL_ERROR_SAVE);
                System.exit(0);
            }
        }

        // to skip that particular line
        bufTempReader.readLine();

        // copy the rest of the file to the temporary file
        while ((tempLineReader = bufTempReader.readLine()) != null)
        {
            bufTempWriter.write(tempLineReader);
            bufTempWriter.newLine();
        }

        bufTempReader.close();
        this._bufFileWriter.close();
        bufTempWriter.close();

        oldFile.delete();
        tempFile.renameTo(oldFile);
        this._bufFileWriter = new BufferedWriter(new FileWriter(tempFile, true));

    }
    catch (IOException io)
    {
        this.informUser(MESSAGE_INTERNAL_ERROR_SAVE);
        return false;
    }
    catch (SecurityException se)
    {
        this.informUser("Access Denied to modify file system in order to save the changes!");
        return false;
    }
    finally
    {
        try
        {
            if (bufTempReader != null)
            {
                bufTempReader.close();
            }
            if (bufTempWriter != null)
            {
                bufTempWriter.close();
            }
        }
        catch (IOException ioe)
        {
            this.informUser(MESSAGE_INTERNAL_ERROR);
            System.exit(0);
        }
    }
    return true;
}

/* a performance enhancement to save multiple unsaved changes at once */
private void writeLocalCopyToFile ()
{
    try
    {
        this._bufFileWriter.close();
        this._bufFileWriter = new BufferedWriter(new FileWriter(new File(this._fileName)));
        Iterator<String> iter = this._localCopyOfFileData.iterator();
        while (iter.hasNext())
        {
            this._bufFileWriter.write(iter.next());
            this._bufFileWriter.newLine();
        }
    }
    catch (IOException ioe)
    {
        ioe.printStackTrace();
    }
    finally
    {
        try
```

```
      {
         this._bufFileWriter.flush();
      }
      catch (IOException ioe)
      {
         ioe.printStackTrace();
      }
   }
}

/* House keeping work */
public void close ()
{
   if (_autoSaver != null)
   {
      try
      {
         // inform the auto saver that the program has ended
         this._hasEnded = true;
         // wait for auto saver to complete its job
         this._autoSaver.join();
      }
      catch (SecurityException se)
      {
         this.informUser(MESSAGE_INTERNAL_ERROR);
      }
      catch (InterruptedException ie)
      {
         ie.printStackTrace();
      }
   }

   this._autoSaver = null;
   this._unsavedChanges.clear();
   this._localCopyOfFileData.clear();
   this._fileName = "";
   this._inputStream.close();
   try
   {
      if (_bufFileWriter != null)
      {
         this._bufFileWriter.close();
      }
   }
   catch (IOException io)
   {
      this.informUser("An error occurred while trying to close the file hander!");
   }
   this._hasNotClosed = false;
}

/* call close in case it wasn't */
public void finalise ()
{
   if (this._hasNotClosed)
   {
      this.close();
   }
}

public static void main (String[] args)
{
   boolean isValidParameter = validateParameter(args);

   if (isValidParameter)
   {
      try
      {
         TextBuddy Textbuddy = new TextBuddy(args[0]);
         Textbuddy.start();
      }
      catch (FileNotFoundException fe)
      {
         out.println("Given file could not be opened for reading/writing.");
      }
      catch (IOException io)
      {
         out.println("An input-output exception occurred while trying to open/read from/write into the file given!");
      }
      catch (SecurityException se)
      {
         out.println("The program not have enough permissions to access/write into the file!");
      }
   }
   else
   {
      out.println("No file given!");
   }
   return;
}

/* Checks if the user has provided a filename to work with */
private static boolean validateParameter (String fileName[])
{
   boolean singleParameterGiven = fileName.length == 1;

   if (singleParameterGiven)
   {
      boolean parameterIsNotEmpty = fileName[0].length() > 0;
      return parameterIsNotEmpty;
   }

   return false;
}

/* An enumeration for all possible commands */
```

```
private enum COMMAND_TYPE
{
   ADD, DELETE, CLEAR, DISPLAY, EXIT, UNKNOWN
}

/* Dissects user commands into the various components */
private class UserCommandInterpretter
{
   public COMMAND_TYPE _command;
   public String _strparam;
   public int _intparam;
   final int VALUE_NOT_SET = -1;
   final int WRONG_VALUE_GIVEN = -2;
   private static final String UNKNOWN_COMMAND_RESPONSE_STRING = "Given word is not a recognised command!";
   private static final String WRONG_PARAMETERS_GIVEN_RESPONSE_STRING = "Given command inexecutable with given parameters!";
   private static final String MESSAGE_INTEGER_NOT_GIVEN = "Given value is not a valid integer!";

   private Scanner _userCommandReader;

   /*
    * Takes a user command and splits it into a commandWord and associated
    * parameters
    */
   public UserCommandInterpretter (String command)
   {
      /* set default values */
      this._strparam = "";
      this._intparam = VALUE_NOT_SET;
      this._command = COMMAND_TYPE.UNKNOWN;

      if (!command.isEmpty())
      {
         this._userCommandReader = new Scanner(command);
         this.dissectUserCommand();
         this._userCommandReader.close();
      }
      else
      {
         this._strparam = UNKNOWN_COMMAND_RESPONSE_STRING;
         this._command = COMMAND_TYPE.UNKNOWN;
         this._intparam = WRONG_VALUE_GIVEN;
      }
   }

   private void dissectUserCommand ()
   {
      String commandWord = _userCommandReader.next().toLowerCase();

      switch (commandWord)
      {
         case "add":
            this.handleAddCommand();
            break;
         case "delete":
            this.handleDeleteCommand();
            break;
         case "display":
            this.handleDisplayCommand();
            break;
         case "clear":
            this.handleClearCommand();
            break;
         case "exit":
            this.handleExitCommand();
            break;
         default:
            this._strparam = UNKNOWN_COMMAND_RESPONSE_STRING;
            this._command = COMMAND_TYPE.UNKNOWN;
            this._intparam = WRONG_VALUE_GIVEN;
            break;
      }
   }

   private void handleExitCommand ()
   {
      this._command = COMMAND_TYPE.EXIT;
   }

   private void handleClearCommand ()
   {
      this._command = COMMAND_TYPE.CLEAR;
   }

   private void handleDisplayCommand ()
   {
      this._command = COMMAND_TYPE.DISPLAY;
   }

   private void handleDeleteCommand ()
   {
      String userParameter;
      this._command = COMMAND_TYPE.DELETE;
      try
      {
         userParameter = _userCommandReader.nextLine().trim();
         this._intparam = Integer.parseInt(userParameter);

      }
      catch (NoSuchElementException e)
      {
         this._strparam = WRONG_PARAMETERS_GIVEN_RESPONSE_STRING;
         this._intparam = WRONG_VALUE_GIVEN;
      }
      catch (NumberFormatException nfe)
      {
         this._intparam = WRONG_VALUE_GIVEN;
```

```
            this._strparam = MESSAGE_INTEGER_NOT_GIVEN;
         }
      }

      private void handleAddCommand ()
      {
         String userParameter;
         try
         {
            this._command = COMMAND_TYPE.ADD;
            userParameter = _userCommandReader.nextLine().trim();
            this._strparam = userParameter;
         }
         catch (NoSuchElementException e)
         {
            this._strparam = "";
         }
      }

      public COMMAND_TYPE getCommand ()
      {
         return this._command;
      }

      public String getStringParameter ()
      {
         return this._strparam;
      }

      public int getIntParameter ()
      {
         return this._intparam;
      }

      public String getErrorString ()
      {
         return this._strparam;
      }
   }// UserCommandInterpretter Class
}// TextBuddy Class
```

TestInput.txt

Test Input I

a
display
add
delete a
delete 1
delete
clear

add Hello World!
add maybe
add this one is a very long comment..!@
display
delete 2
display
delete 3
display
clear
display
exit

ExpectedOutput.txt

```
Welcome to TextBuddy. test.txt is ready for use
command: Given word is not a recognised command!
command: test.txt is empty
command: added to test.txt: ""
```

```
command: Given value is not a valid integer!
command: deleted from test.txt: ""
command: Given command inexecutable with given parameters!
command: all content deleted from test.txt
command: Given word is not a recognised command!
command: added to test.txt: "Hello World!"
command: added to test.txt: "maybe"
command: added to test.txt: "this one is a very long comment..!@"
command: 1. Hello World!
2. maybe
3. this one is a very long comment..!@
command: deleted from test.txt: "maybe"
command: 1. Hello World!
2. this one is a very long comment..!@
command: Given index is out-of-range of the available number of lines!
command: 1. Hello World!
2. this one is a very long comment..!@
command: all content deleted from test.txt
command: test.txt is empty
command:
```

TestInput2.txt

The input file is too long to be pasted here. Please refer to testinput2.txt for the actual input commands

ExpectedOutput2.txt

The output file is too long to paste here. Please refer to testinput2.txt for the actual output commands