ECHO

TODO MANAGER

DEVELOPER GUIDE

Contents

Preface

This documentation walks you to understanding the architecture on which Echo is built and the way various components work together so that you can help improve and extend Echo. It is recommended that you are familiar with the common Java APIs including the Swing framework.

Credits

Here are the development team of Echo:

David Heryanto	Liu Yuanrui	Lokman Hakim	Yuan Shuai
Lead GUI developer, Tester, Documenter	Lead documenter, GUI developer, schedule watcher	Team leader, lead programmer, cleanup crew	Team organizer, tester, documenter, lead cleanup

1. Overview

Echo is a portable application that helps people manage their daily tasks. As a to-do list, the basic functions are create, edit, delete and search. Graphical User Interface (GUI) with natural language input and autocomplete are extra features. The main architecture components are GUI, logic handler and storage. Currently, in the progress of Product V0.1 releasing, most effort is made in the implementing these three parts based on basic functions in the integrated IDE of Eclipse. Therefore as the incoming developer, your jobs are keeping working on basic functions to make improvement and completing the extra features of ECHO subsequently. In the developing process, the main challenge will be the coordination of the three parts of architecture.

2. Architecture

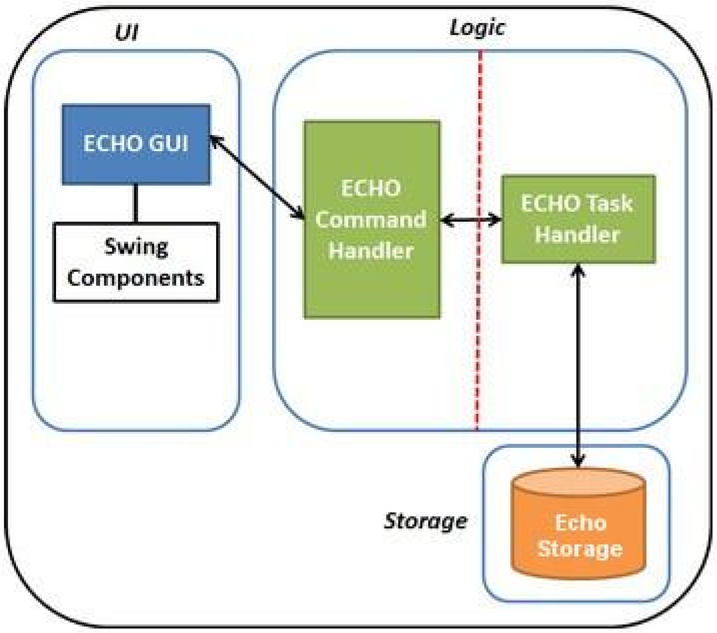


Fig. 1. Architecture design of ECHO

Fig 1 shows the high level architecture of Echo. EchoGUI, as the front end component of Echo, interacts with the user while EchoCommandHandler and EchoTaskHandler are the back end component that manages user's tasks. EchoStorage handles the storage of user's tasks into a persistent file on the user's computer.

The arrows in Fig 1 show how each component above interact with another. EchoCommandHandler acts as a facade[1] that bridge the interaction between EchoGUI and EchoTaskHandler. User's input passed through the user interface is sent to EchoCommandHandler where the input is parsed and the

[1] facade or façade is a French word that means 'front of a building'

relevant method in EchoTaskHandler is called. Upon handling the command, EchoTaskHandler will call EchoStorage to update the persistent file. This is to minimize the risk of losing any unsaved data.

3. Software Design

This section explains how different components in the architecture are designed and interact with the other components. Fig 2 is the class diagram of Echo which provides a brief outlook of the whole software design.

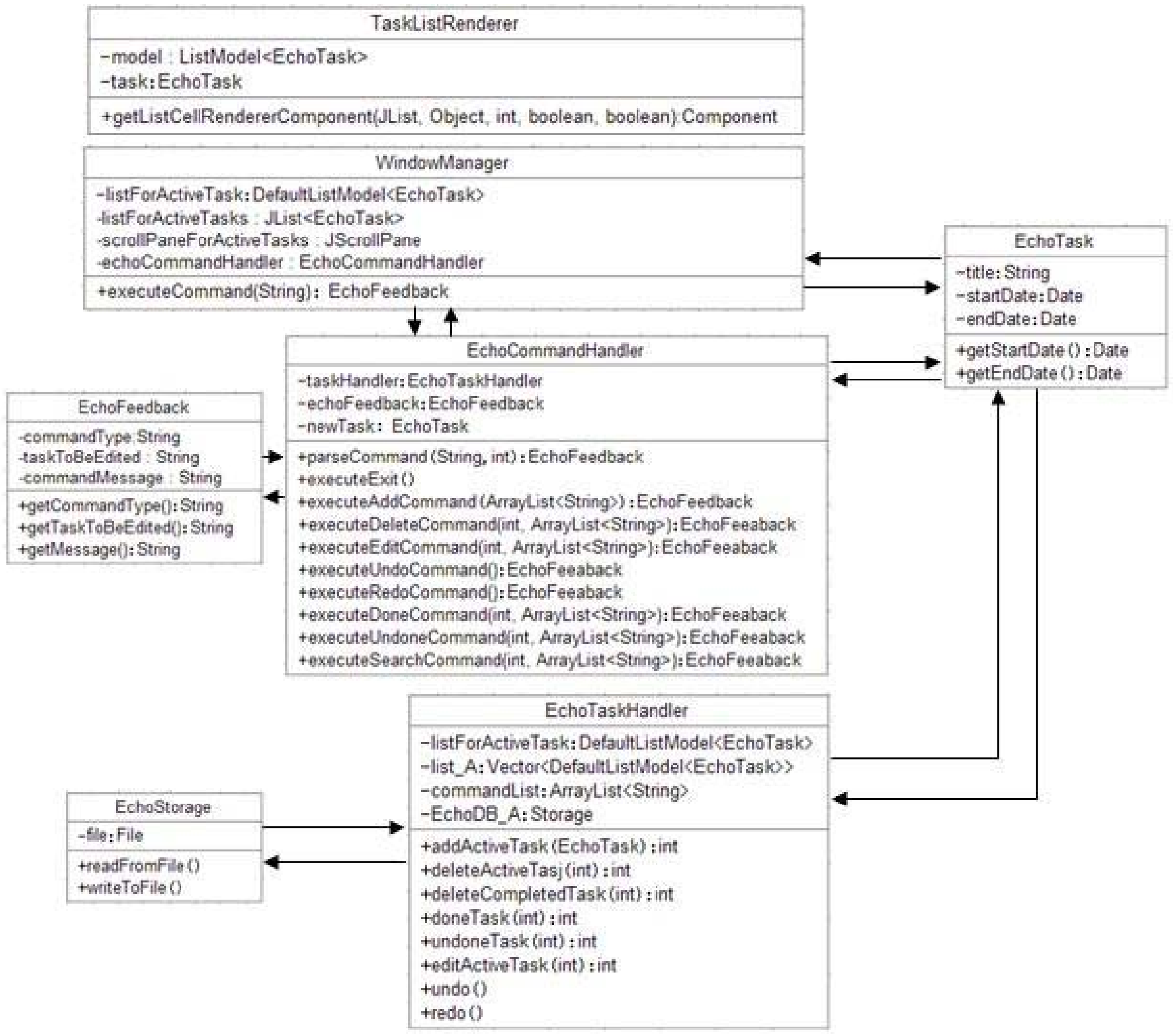

TaskListRenderer
-model : ListModel<EchoTask>
-task: EchoTask
+getListCellRendererComponent(JList, Object, int, boolean, boolean): Component
WindowManager
-listForActiveTask: DefaultListModel<EchoTask>
-listForActiveTasks : JList<EchoTask>
-scrollPaneForActiveTasks : JScrollPane
-echoCommandHandler : EchoCommandHandler
+executeCommand(String): EchoFeedback
EchoTask
-title: String
-startDate: Date
-endDate: Date
+getStartDate () : Date
+getEndDate () : Date
EchoCommandHandler
-taskHandler: EchoTaskHandler
-echoFeedback: EchoFeedback
-newTask: EchoTask
+parseCommand (String, int) : EchoFeedback
+executeExit ()
+executeAddCommand (ArrayList<String>) : EchoFeedback
+executeDeleteCommand(int, ArrayList<String>): EchoFeeback
+executeEditCommand(int, ArrayList<String>): EchoFeeback
+executeUndoCommand(): EchoFeeback
+executeRedoCommand(): EchoFeeback
+executeDoneCommand(int, ArrayList<String>): EchoFeeback
+executeUndoneCommand(int, ArrayList<String>): EchoFeeback
+executeSearchCommand(int, ArrayList<String>): EchoFeeback
EchoFeedback
-commandType:String
-taskToBeEdited : String
-commandMessage : String
+getCommandType(): String
+getTaskToBeEdited(): String
+getMessage(): String
EchoTaskHandler
-listForActiveTask: DefaultListModel<EchoTask>
-list_A: Vector<DefaultListModel<EchoTask>>
-commandList: ArrayList<String>
-EchoDB_A: Storage
+addActiveTask (EchoTask) : int
+deleteActiveTasj (int) : int
+deleteCompletedTask (int) : int
+doneTask (int) : int
+undoneTask (int) : int
+editActiveTask (int) : int
+undo ()
+redo ()
EchoStorage
-file: File
+readFromFile ()
+writeToFile ()

Fig. 2. Class Diagram of Echo

3.1 UI

The UI class implements Java Swing API. The user interface is presented in a JFrame object which has the following structure:

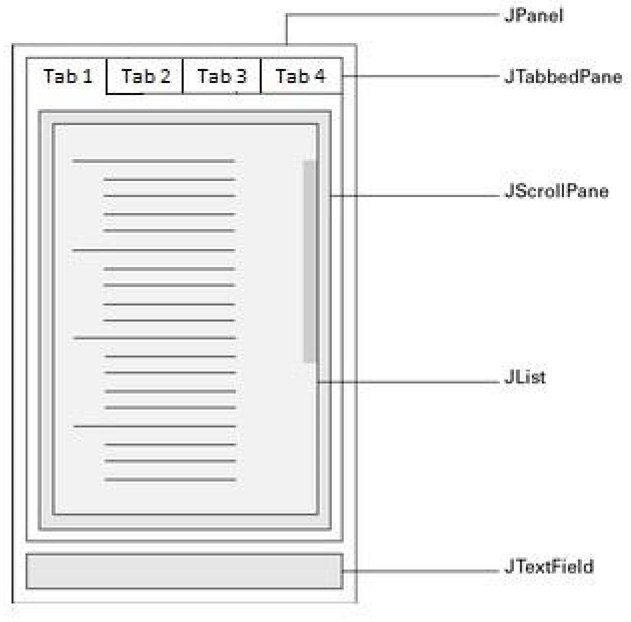


Fig. 3. Structure of components inside JFrame

JTabbedPane object consists of different tabs which the user can select to view different content presented inside the JList object. The user interacts with Echo by typing command into the JTextField which will send the command to the Logic component.

3.1.1. Presenting the list of task in the JList

In terms to the model-view-controller (MVC) architecture which separates the representation of information from the user's interaction with it, JList represent the view component of the MVC while the model that consists of the tasks list is represented by a DefaultListModel object. The controller that mediates input and converts it to commands for the model or view is represented by the EchoCommandHandler object.

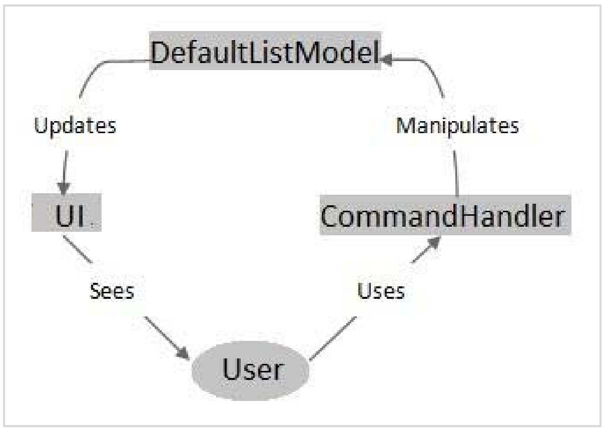


Fig. 4. Sketch of MVC

3.1.2. Processing user command

As stated in section 3.1.1., user commands are processed by EchoCommandler which acts as the controller that modifies the DefaultListModel. Changes to DefaultListModel will fire an event so that JList will update its view.

3.2. CommandHandler

The EchoCommandHandler class sits between the back end classes and the user interface class such that all access to the components passes through it. Meanwhile, it parses the string of user input that user interface passes to it into different parameters according to the user command type. After that, EchoCommandHandler passes all the parameters to the proper methods in EchoTaskHandler class.

3.2.1. Allow access without exposing internal details of component

As shown in Section 2, Echo architecture, EchoCommandHandler acts as a façade. It sits between the internal components and the user interface such that all access to the components passes through the EchoCommandHandler class. For example, EchoCommandHandler has a method that gets the DefaultListModel inside EchoTaskHandler, and passes it to the UI class without modifying it. Figure 5 shows the application of EchoCommandHandler class.

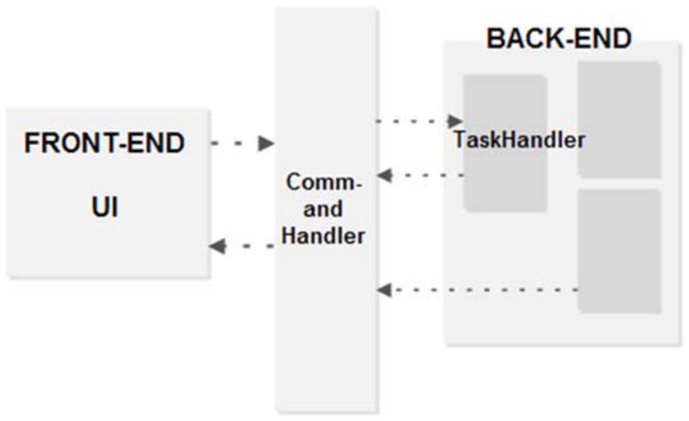


Fig. 5. Class diagram of application of CommandHandler class

3.2.2. Parsing user command

Fig 6 shows the mechanisms of creating an active task in EchoCommandHandler. In the process, EchoCommandHandler separates each parameter inside class CommandHandler and call the corresponding function in TaskHandler.

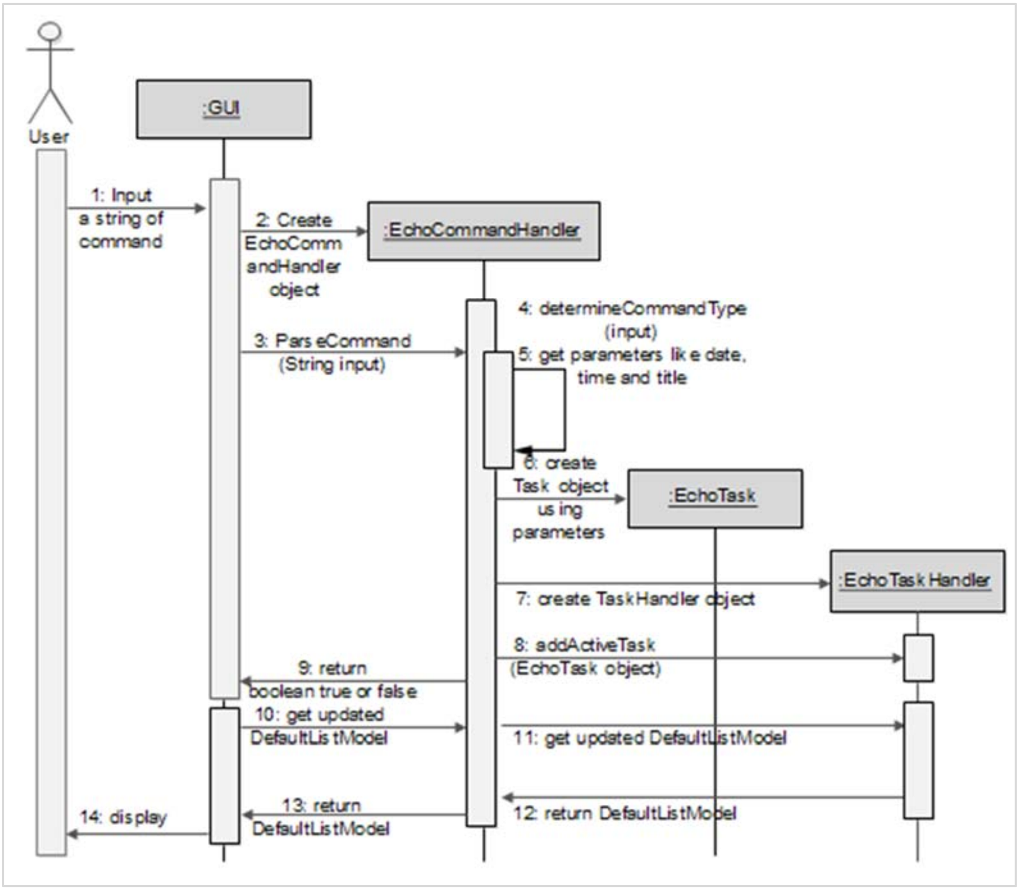

User
:GUI
:EchoCommandHandler
:EchoTask
:EchoTaskHandler
1: Input a string of command
2: Create EchoCommandHandler object
3: ParseCommand (String input)
4: determineCommandType (input)
5: get parameters like date, time and title
6: create Task object using parameters
7: create TaskHandler object
8: addActiveTask (EchoTask object)
9: return boolean true or false
10: get updated DefaultListModel
11: get updated DefaultListModel
12: return DefaultListModel
13: return DefaultListModel
14: display

Fig. 6. Sequence diagram for "create" a task

For parsing the String into separate parameters, the EchoCommandHandler class has following methods:

Visibility	Method	Description
public	`EchoFeedback parseCommand(String inputString, int tabIndex)`	Parses the input command string, return an echoFeedback object
public	`CommandType determineCommandType(String inputString)`	Determines command type
public	`getAddParameters(ArrayList<String> inputArray)`	Gets the parameters of title, start date and end date in different types of task inputs
public	`int getEditParameter(ArrayList<String> inputArray)`	Returns the parameter user enters in edit command
public	`String getSearchParameters(ArrayList<String> inputArray)`	Returns the parameter user enters in search command
public	`ArrayList<Integer> getTaskIndices(ArrayList<String> inputArray)`	Returns the arrayList of integer indexes user enters in each command, like "delete", "done" and "undone"
public	`Date getDate(ArrayList<String> dateArray)`	Gets the valid date including date and time from the input
public	`int getTitleEndIndex(ArrayList<String> inputArray)`	Returns the end index of the task title in order to get task title
public	`boolean isDate(String str)`	Checks whether the string of parameter is date in a certain format
public	`boolean isTime(String str)`	Checks whether the string of parameter is time in a certain format

3.3. TaskHandler

The TaskHandler class is responsible for arranging all the various types of command (e.g. add, delete, edit, etc.) which has been processed by the CommandHandler class and make the data ready for storage in the EchoStorage. The various EchoTask objects will be stored inside DefaultListModels which will be used either for display at the UI or for the updating of the storage. The essential functions are listed below.

3.3.1. EchoTask Constructors

The constructors featured in EchoTask allows for either floating tasks without date and time or normal tasks with date or time. For example:

```
new EchoTask();
new EchoTask(title);
new EchoTask(title,startDate,endDate);
```

3.3.2. EchoStorage Constructor

EchoTaskHandler constructs different EchoStorage objects for active tasks, completed tasks and due tasks. They are shown respectively as below:

```
EchoStorage echoDB_a = new EchoStorage(Active);
EchoStorage echoDB_d = new EchoStorage(Due);
EchoStorage echoDB_c = new EchoStorage(Completed);
```

3.3.3. Manipulating commands

After called by EchoCommandHandler, methods in taskHandler do corresponding actions to finish the command. DefaultListModel<EchoTask> will be changed and returned to EchoCommandHandler and EchoGUI. Meanwhile, the storage will be updated. By now, available methods for executing commands are as below.

Visibility	Method	Description
`public`	`int addActiveTask(EchoTask newTask)`	To add new task in listModelForActiveTasks
`public`	`int deleteActiveTask(int index)`	To delete unwanted task(s) in listModelForActiveTasks
`public`	`int deleteCompletedTask(int index)`	To delete unwanted task(s) in listModelForCompletedTasks

`public`	`int deleteSearchedTask(int index)`	In search panel, deletes unwanted task(s) in listModleForActiveTasks and listModelForSearchedTasks at the same time.
`public`	`int done(int index)`	Moves task in listModelForActiveTasks to listModelForCompletedTasks
`public`	`int undone(int index)`	Moves task in listModelForCompletedTasks to listModelForActiveTasks
`public`	`String editActiveTask(int index)`	Edits tasks in listModelForActiveTasks
`public`	`int searchByTitle(String str)`	Searches existing tasks by keyword in the title
`public`	`undo()`	undo the last command
`public`	`redo()`	redo the last command

3.4. Storage

To organize three different kinds of tasks as active tasks, completed tasks and due tasks, they are recorded in three comma-separated values (csv) files EchoDB_A.csv, EchoDB_C.csv, EchoDB_D.csv accordingly.

3.4.1. readFromFile

After starting Echo, EchoTaskHandler will initialize three DefaultListModles of EchoTask for active tasks, completed tasks and due tasks by calling method readFromFile in EchoStorage accordingly.

In readFromFile, it first reads Strings from csv files and analyze them one by one to get sets of parameters. Then it uses each set of parameters construct each EchoTask in the DefaultListModel. After reading the whole csv file, it returns expected DefaultListModel to initialize the DefaultListModel in EchoTaskHandler.

3.4.2. writeToFile[2]

Echo updates the storage when a command is made. This ensure that each time new change will be recorded in storage securely. After the user presses enter for one

[2] This will be the reverse of "readFromFile()" process.

command, the storage csv file will update due to the changed DefaultListModel. Each row in the file represents an EchoTask. The format of the row would be:

title/,startDate/,endDate

(startDate/endDate:dd/mm/yyyy HH:mm)

4. Code Examples

4.1. Receive user input from UI

```
//in WindowManager Class
public EchoFeedback executeCommand(String input) {
     /*
     * currentTab is for echoCommandHandler to decide on the appropriate actions
     * e.g. if user is currently viewing 'completed' tab, then 'done' command
     * is not really appropriate
     */
     int currentTab = tabs.getSelectedIndex();
     EchoFeedback feedback = echoCommandHandler.parseCommand(input, currentTab);
     return feedback;}
```

4.2. Parse user input and execute it

```
//in EchoCommandHandler Class
public EchoFeedback parseCommand(String inputString, int tabIndex) {
      EchoFeedback echoFeedback = null;
      // added a check for empty string, 11/7/2012
      if (inputString.length() < 1) {
             echoFeedback = new EchoFeedback("noCommand", ERROR_EMPTY_STRING,
                                    false);
             return echoFeedback;
             }
      assert (tabIndex >= 0 && tabIndex <= 4);
      ArrayList<String> inputArray = tokenizeStrBySpace(inputString);
      CommandType commandType = determineCommandType(inputArray.get(0));
      // log a message at INFO level
      logger.log(Level.INFO, "going to start parsing");
      try {
             switch (commandType) {
                    case EXIT:
                        executeExitCommand();
                        break;
                case ADD:
                        echoFeedback = executeAddCommand(inputArray);
                        break;
                case DELETE:
                        echoFeedback = executeDeleteCommand(tabIndex, inputArray);
                        break;
                case EDIT:
                        echoFeedback = executeEditCommand(tabIndex, inputArray);
                        break;
                case UNDO:
                        echoFeedback = executeUndoCommand();
                        break;
                case REDO:
                        echoFeedback = executeRedoCommand();
                        break;
                case DONE:
                        echoFeedback = executeDoneCommand(tabIndex, inputArray);
                        break;
                case UNDONE:
                        echoFeedback = executeUndoneCommand(tabIndex, inputArray);
                        break;
                case SEARCH:
                        echoFeedback = executeSearchCommand(tabIndex, inputArray);
                        break;
                default:
                        echoFeedback = new EchoFeedback("",
                                        INVALID_COMMAND_ERROR_MESSAGE, false);
                }
        } catch (Exception e) {
                // Log a message at WARNING level
                logger.log(Level.WARNING, "parsing error", e);
        }
    logger.log(Level.INFO, "end of parsing");
    return echoFeedback;}
```

4.3 Handling a due task

```
//in EchoTaskHandler Class
public ActionListener getActionListener() {
        ActionListener dueTaskListener = new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
                if (!listModelForActiveTasks.isEmpty()) {
                Date currentDate = new Date();
                // past
                for (int i = 0; i < listModelForActiveTasks.size(); i++) {
                        if (listModelForActiveTasks.get(i).getStartDate() != null)
                        {
                                if(currentDate.compareTo(listModelForActiveTasks.get
                                (i).getEndDate()) > 0) {
                                        EchoTask dueTask =
                                        listModelForActiveTasks.get(i);
                                        deleteActiveTask(i);
                                        listModelForDueTasks.addElement(dueTask);}}}}
                                        }
        };
        echoDB_d.writeToFile(listModelForDueTasks);
        return dueTaskListener;}
```

5. Application Programming Interface

Listed below are the essential Application Programming Interfaces (API) contained within the different components in Echo, described in full detail in this section.

5.1. CommandHandler

```
public void parseString(String inputString)
```

Parameters: inputString - the input String typed by the user

Separates user input command into parts, and indicate each part as either command name, task title, task time or task date, then call different method in taskHandler to handle each command.

5.2. TaskHandler

```
public int addActiveTask(EchoTask newTask)
```

Parameters: EchoTask - a class designed to handle all the data in a task

This method basically adds the desired task to the listModelForActiveTasks component in EchoTaskHandler.

`public int deleteActiveTask(int index)`

Parameters: index - the index of the EchoTask in the listModelForActiveTasks component.

This method basically deletes the desired task from the listModelForActivieTasks component in EchoTaskHandler using the index given by the User.

`public int searchTaskByTitle(String str)`

Parameters: String- the title of target EchoTask.

This method searches for the desired task from the listModelForActiveTasks in EchoTaskHandler using the title given by the User.

`public String editAvtiveTask(int index)`

Parameters: index - the index of the EchoTask in the listModelForActiveTasks component.

This method deletes the old task in the listModelForActiveTasks and waits the user to enter edited tasks as adding new active task.

`public int doneTask(int index)`

Parameters: index - the index of the EchoTask in the listModelForActiveTasks component.

This method moves just completed task in the listModelForActiveTasks to listModelForCompletedTasks.

`public int undoneTask(int index)`

Parameters: index - the index of the EchoTask in the listModelForCompletedTasks component.

This method undone completed task in listModelForCompletedTasks and move it back to listModelForActiveTasks.

5.3. Storage

`private DefaultListModel<EchoTask> readFromFile()`

Parameters: null

Reads the data from the ECHO Storage.

`private void writeToFile(DefaultListModel<EchoTask>)`

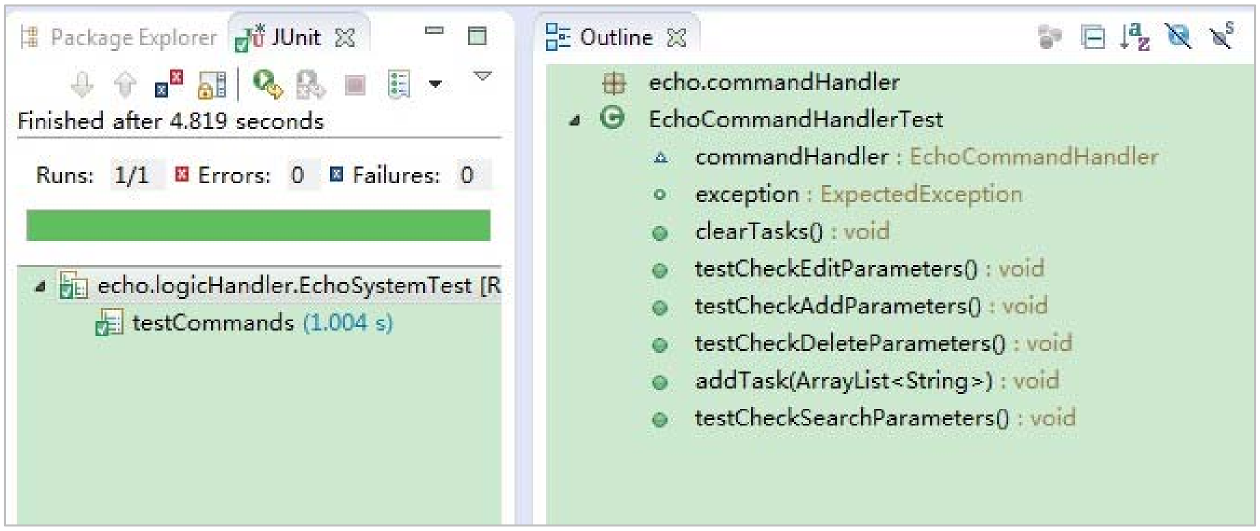
Parameters: DefaultListModel<EchoTask>-the list of tasks to be copied by the storage.

Writes (Saves) the data from ECHO and transfers it to ECHO Storage.

6. Testing Instructions

6.1 Unit Testing

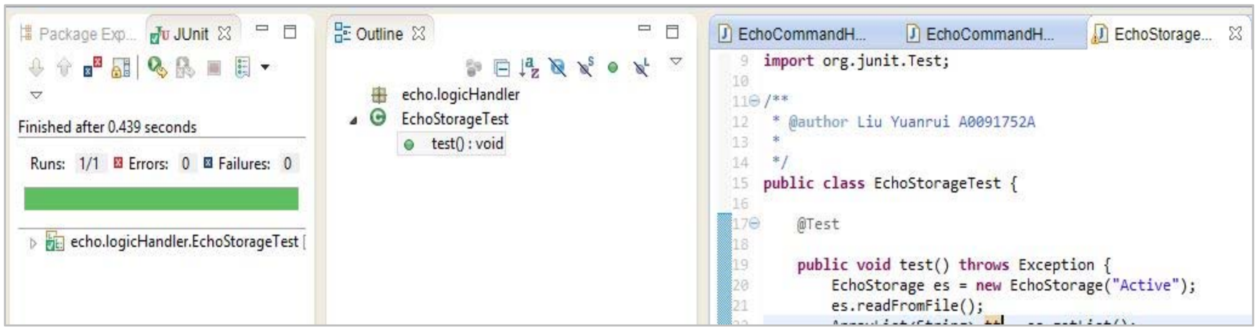
Echo uses JUnit testing framework to automate the testing process. You can run the respective test classes every time you make changes to the code to ensure that your code runs properly.



```
Finished after 4.819 seconds
Runs: 1/1   Errors: 0   Failures: 0
echo.logicHandler.EchoSystemTest [R
    testCommands (1.004 s)
```

```
echo.commandHandler
EchoCommandHandlerTest
    commandHandler : EchoCommandHandler
    exception : ExpectedException
    clearTasks() : void
    testCheckEditParameters() : void
    testCheckAddParameters() : void
    testCheckDeleteParameters() : void
    addTask(ArrayList<String>) : void
    testCheckSearchParameters() : void
```

Fig. 7. JUnit testing for EchoCommandHandler class



```
Finished after 0.439 seconds
Runs: 1/1   Errors: 0   Failures: 0
echo.logicHandler.EchoStorageTest [
```

```
echo.logicHandler
EchoStorageTest
    test() : void
```

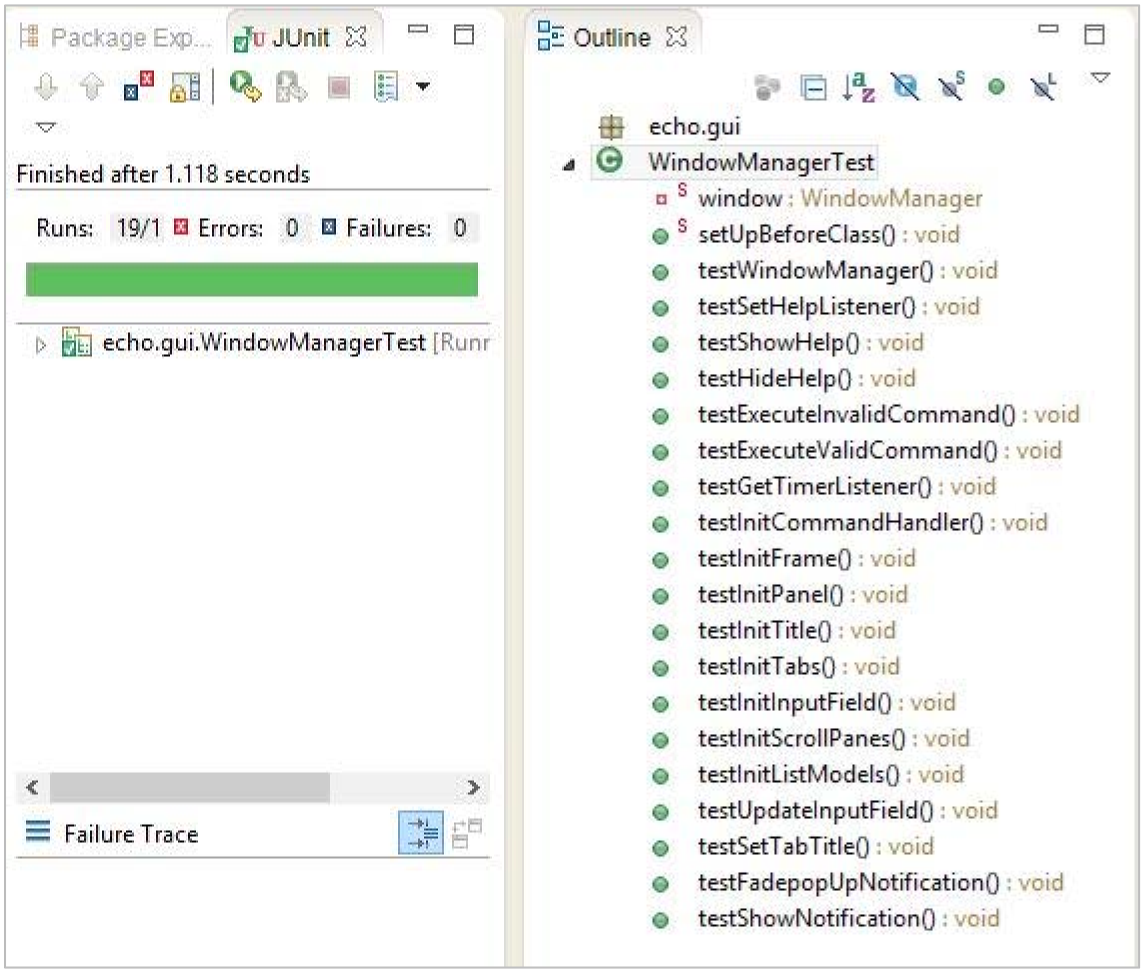
```java
import org.junit.Test;

/**
 * @author Liu Yuanrui A0091752A
 *
 */
public class EchoStorageTest {

    @Test

    public void test() throws Exception {
        EchoStorage es = new EchoStorage("Active");
        es.readFromFile();
```

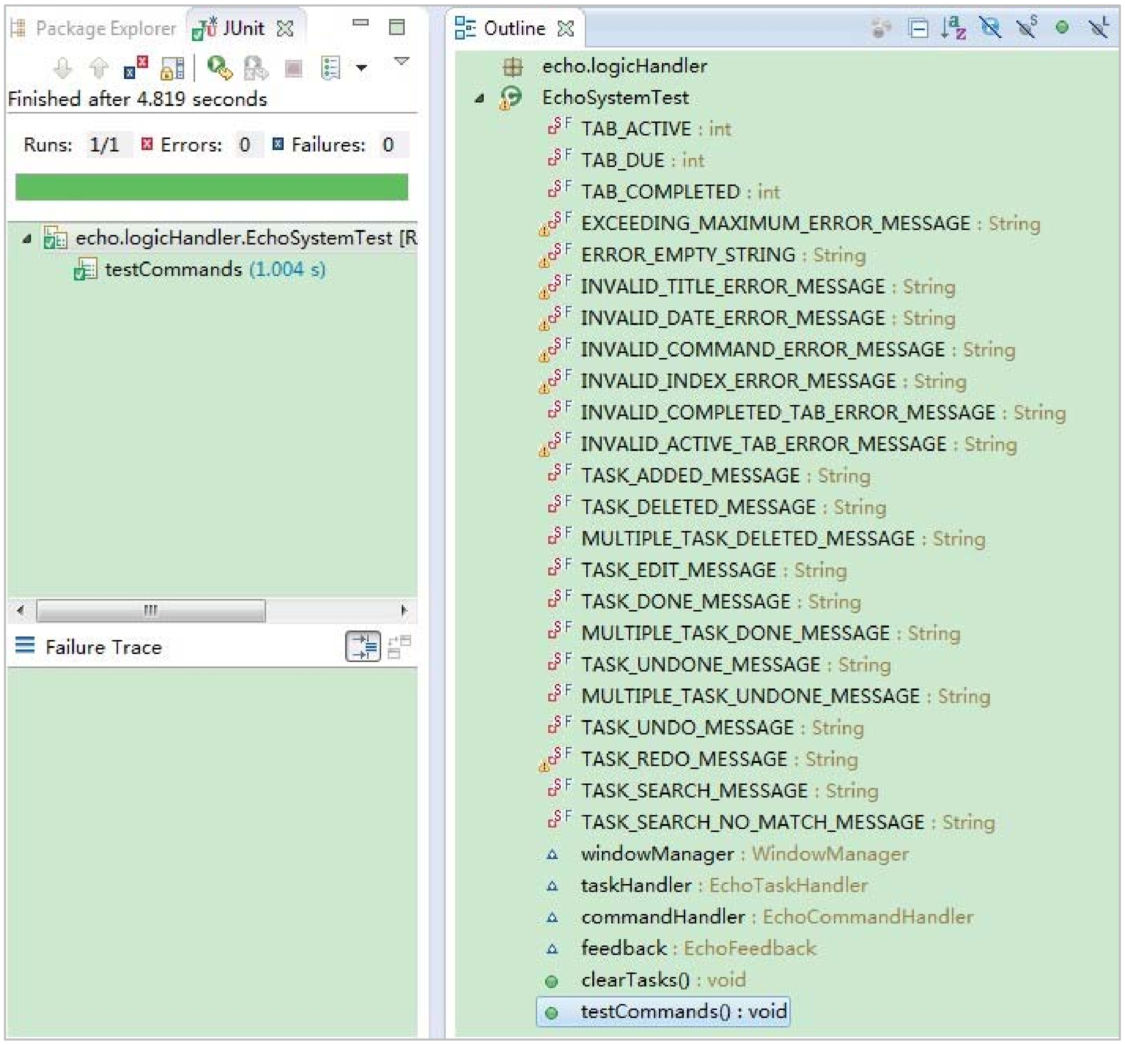
Fig. 8. JUnit testing for EchoStorage class



```
Finished after 1.118 seconds
Runs: 19/1   Errors: 0   Failures: 0
echo.gui.WindowManagerTest [Runr
Failure Trace
```

```
echo.gui
WindowManagerTest
  window : WindowManager
  setUpBeforeClass() : void
  testWindowManager() : void
  testSetHelpListener() : void
  testShowHelp() : void
  testHideHelp() : void
  testExecuteInvalidCommand() : void
  testExecuteValidCommand() : void
  testGetTimerListener() : void
  testInitCommandHandler() : void
  testInitFrame() : void
  testInitPanel() : void
  testInitTitle() : void
  testInitTabs() : void
  testInitInputField() : void
  testInitScrollPanes() : void
  testInitListModels() : void
  testUpdateInputField() : void
  testSetTabTitle() : void
  testFadepopUpNotification() : void
  testShowNotification() : void
```

Fig. 9. JUnit testing for WindowManager class



Finished after 4.819 seconds

Runs: 1/1 Errors: 0 Failures: 0

echo.logicHandler.EchoSystemTest [R
testCommands (1.004 s)

echo.logicHandler
EchoSystemTest
TAB_ACTIVE : int
TAB_DUE : int
TAB_COMPLETED : int
EXCEEDING_MAXIMUM_ERROR_MESSAGE : String
ERROR_EMPTY_STRING : String
INVALID_TITLE_ERROR_MESSAGE : String
INVALID_DATE_ERROR_MESSAGE : String
INVALID_COMMAND_ERROR_MESSAGE : String
INVALID_INDEX_ERROR_MESSAGE : String
INVALID_COMPLETED_TAB_ERROR_MESSAGE : String
INVALID_ACTIVE_TAB_ERROR_MESSAGE : String
TASK_ADDED_MESSAGE : String
TASK_DELETED_MESSAGE : String
MULTIPLE_TASK_DELETED_MESSAGE : String
TASK_EDIT_MESSAGE : String
TASK_DONE_MESSAGE : String
MULTIPLE_TASK_DONE_MESSAGE : String
TASK_UNDONE_MESSAGE : String
MULTIPLE_TASK_UNDONE_MESSAGE : String
TASK_UNDO_MESSAGE : String
TASK_REDO_MESSAGE : String
TASK_SEARCH_MESSAGE : String
TASK_SEARCH_NO_MATCH_MESSAGE : String
windowManager : WindowManager
taskHandler : EchoTaskHandler
commandHandler : EchoCommandHandler
feedback : EchoFeedback
clearTasks() : void
testCommands() : void

Fig. 10. System testing for Echo

7. Known Issues and future development

By now, for the product V0.5 of Echo, all planned functions are implemented as expected and are ready to be released. The future development will focus on improving user-friendly design and increasing practical functions.

After summarizing, the main tasks in further development of Echo are shown below.

Future tasks	***Developing plan***
Sync with Google Calendar	Connect with Google Calendar to increase the usability of Echo.
Infinite undo and redo function	Future Echo will support infinite times undo and redo command
Auto complete	Auto complete when user is searching by keywords. This function will be implemented inside Echo logicHandler. Java language library will be used.
Multi-platform Development	Since Echo is a java software, it will be developed in different platforms in the future, such as Linux and IOS.

8. Change log

Echo V0.5

- Support more date formats for user input

- All unit testings have been done

- System testing has been done

Echo V0.4

- “delete” function can delete more than one task

- Move EchoStorage to a separated class from EchoTaskHandler

- Further implemented all the functions

Echo V0.3

- Add basic features including “mark as done” and “mark as undone”, as well as a basic “undo”

Echo V0.2

- Modified the UI of Echo. Tasks are now shown in tabbed panes rather than in one big panel
- EchoTask object is now stored in a DefaultListModel instead of a Vector
- Testing instructions in the developer guide are further elaborated.

Echo V0.1

- Initial release of Echo with three main features: “add”, “delete” and “edit”.