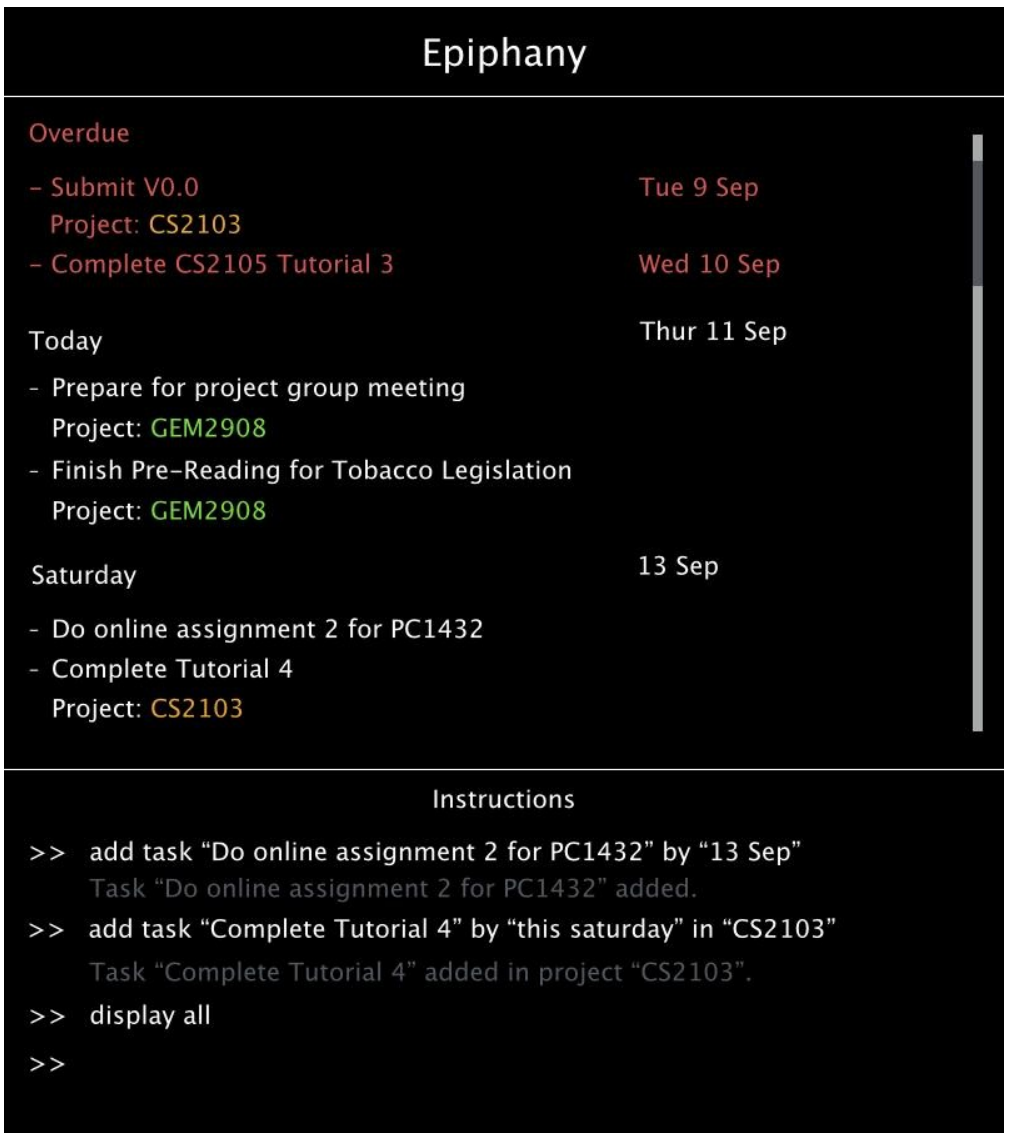
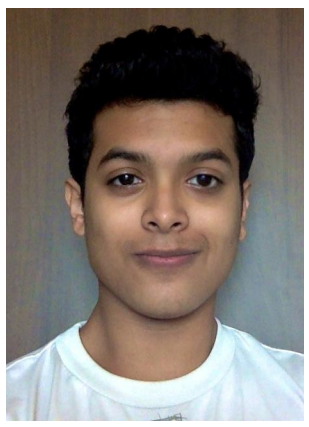
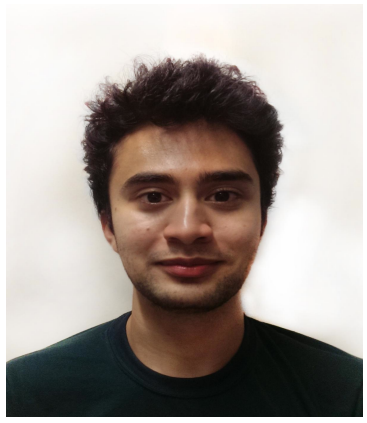
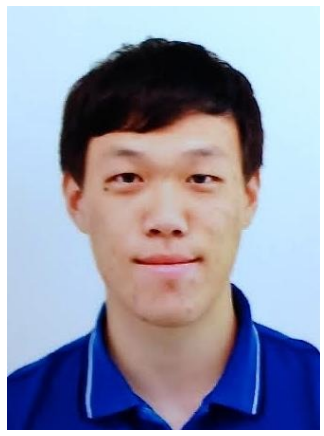
Epiphany

UI Sketch



Supervisor: Shawn Lee

Extra feature: Google Calendar Integration

 Amit Team Lead/Git Expert	 Abdulla Integration/ UX Design	 Moazzam Code Quality	 Wei Yang Testing

User Guide

Overview

Epiphany is a great and simple way to manage your daily tasks. Compared to conventional GTD software, Epiphany is intended to be lightweight and allows you to input your tasks in a simple and straightforward manner.

Though Epiphany works via the command line, you don't have to learn verbose commands to manipulate your tasks. All you have to do is type in a task along with the date(if any) and it gets added to your list of upcoming tasks! We believe this makes management of simple deadlines much easier!

Using Epiphany

There are several features supported in Epiphany that all work cohesively to help you manage your tasks better. Along with basic tasks, Epiphany also allows you to sort your tasks into projects for easier management.

With Epiphany, you can now

- *add, edit & delete tasks, with support for timed tasks.*
- *search for a task, even if you type only a fraction of the task description*
- *display tasks on a given day or even by project*
- *view all your projects*
- *sync your data with Google Calendar.*

Here are some examples to get you going:

Adding Tasks

Adding tasks can be done with a simple intuitive command which is close to natural English. Here the quotes act as delimiters and indicate where the content to be added begins and ends.

Input: add "taskDescription"
Description: adds a task without a deadline attached to it - a floating task
Example: add "Save all of mankind"

User Guide

Input: add “taskDescription” by “dueDate”
Description: adds a task with a deadline attached to it.
Example: add “Prepare for project meeting” by “this Friday”

Input: add “taskDescription” #projectName
Description: adds a task without a deadline attached to it, to a specific project.
Example: add “Save all of mankind” #justEverydayThings

Input: add “taskDescription” by “dueDate” #projectName
Description: adds a task with a deadline attached to it.
Example: add “Prepare for project meeting” by “this Friday” #GEM2908

Editing and Deleting Tasks

Input: edit <task description> to <new task description>
Description: update your task description with the above format.
Example: edit “Meet Hannah by this Friday” to “Meet Hannah by this Saturday”

Input: delete <task description>
Description: In the placeholder, you can type part of the task description and Epiphany will pull up a list of tasks that match closely. After which, you then can choose which of those tasks to delete.
Example: delete friends
delete <index>

Input: <task description> is done
Description: Given the task description, you can mark it as done.
Example: Assignment 4 is done

Input: change deadline of <task description> to <new deadline>
to Description: adds a task with a new deadline attached to it.
Example: change deadline of “Prepare for project meeting” to 16 December.

User Guide

Displaying Tasks

Input: display all
Description: displays all the tasks that the user has added which are sorted according to date. This takes the form of an infinitely scrollable list.

Input: display today
Description: displays all the tasks that the user has due today. This screen would also display all the overdue tasks. This would also be the default view in the GUI.

Input: display ongoing tasks
Description: displays all tasks that are currently ongoing.

Input: display #projectName
Description: displays all the tasks that the user has added under a specific project.
Example: display #CS2103

Input: display “date”
Description: displays all tasks due on a specific date.
Example: display “15 Sep”

Project Management

Input: create new project #projectName
Description: You can use this syntax to create a new project(list). You can later add tasks to this project using the syntax described in the adding tasks section.
Example: create new project #PC1432

Input: update #projectName to #newProjectName
Description: You can use this syntax to update an existing project.
Example: update project #PC1432 to #PC1431

User Guide

Search

Input: search “keyword/phrase”

Description: predictively searches all task names (both inside and outside of projects) for this keyword. Not case sensitive. Displays all of these in the output terminal afterwards.

Example: search “essay”

Sync

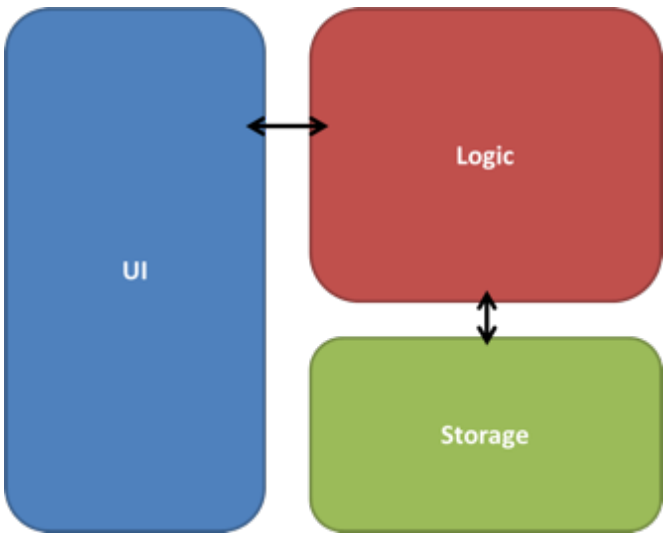
Input: sync

Description: Synchronizes all your tasks to Google Calendar.

Developer Guide

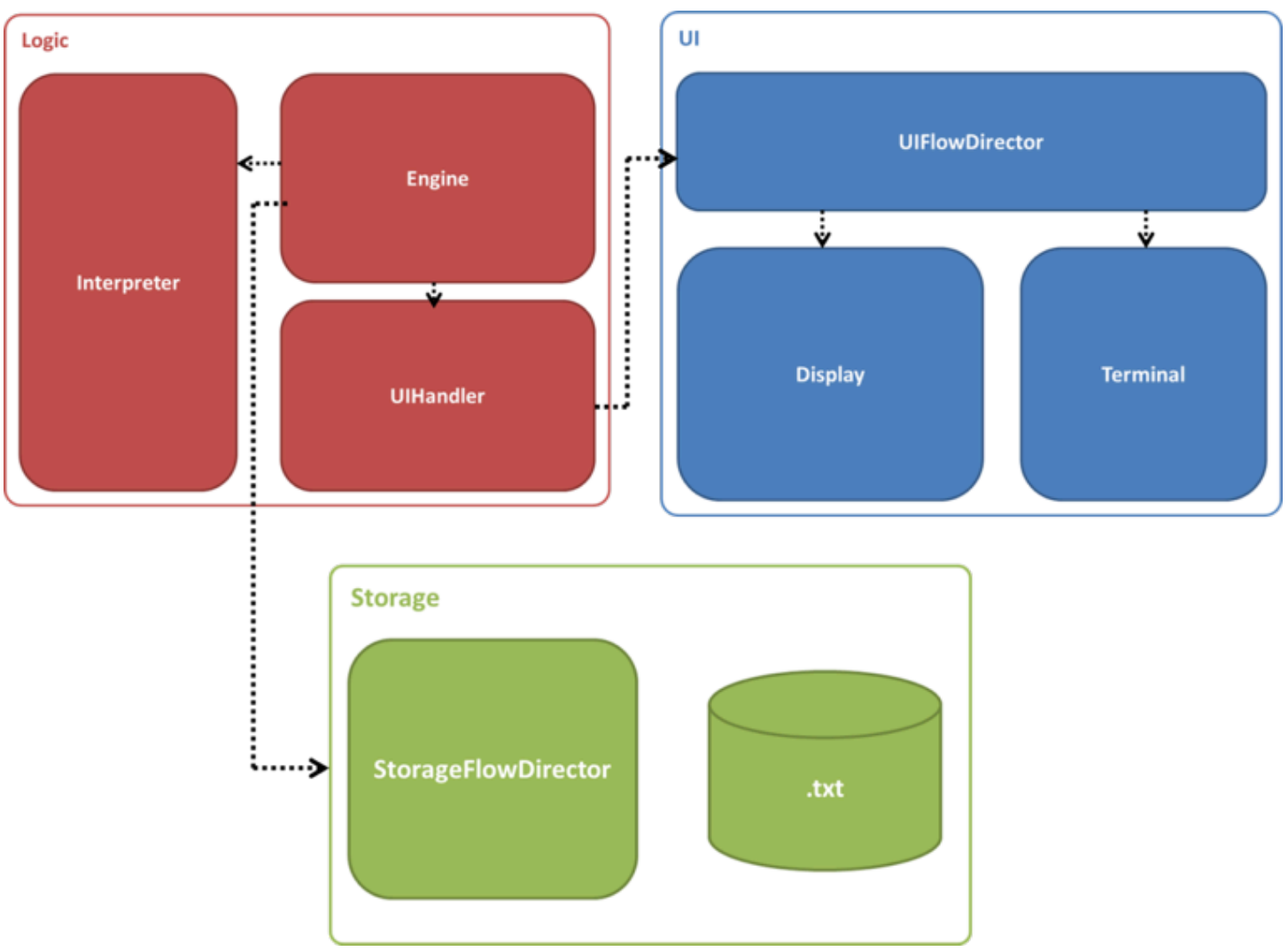
Architecture overview

Epiphany is written in Java. Given above is a high level overview of its main components.



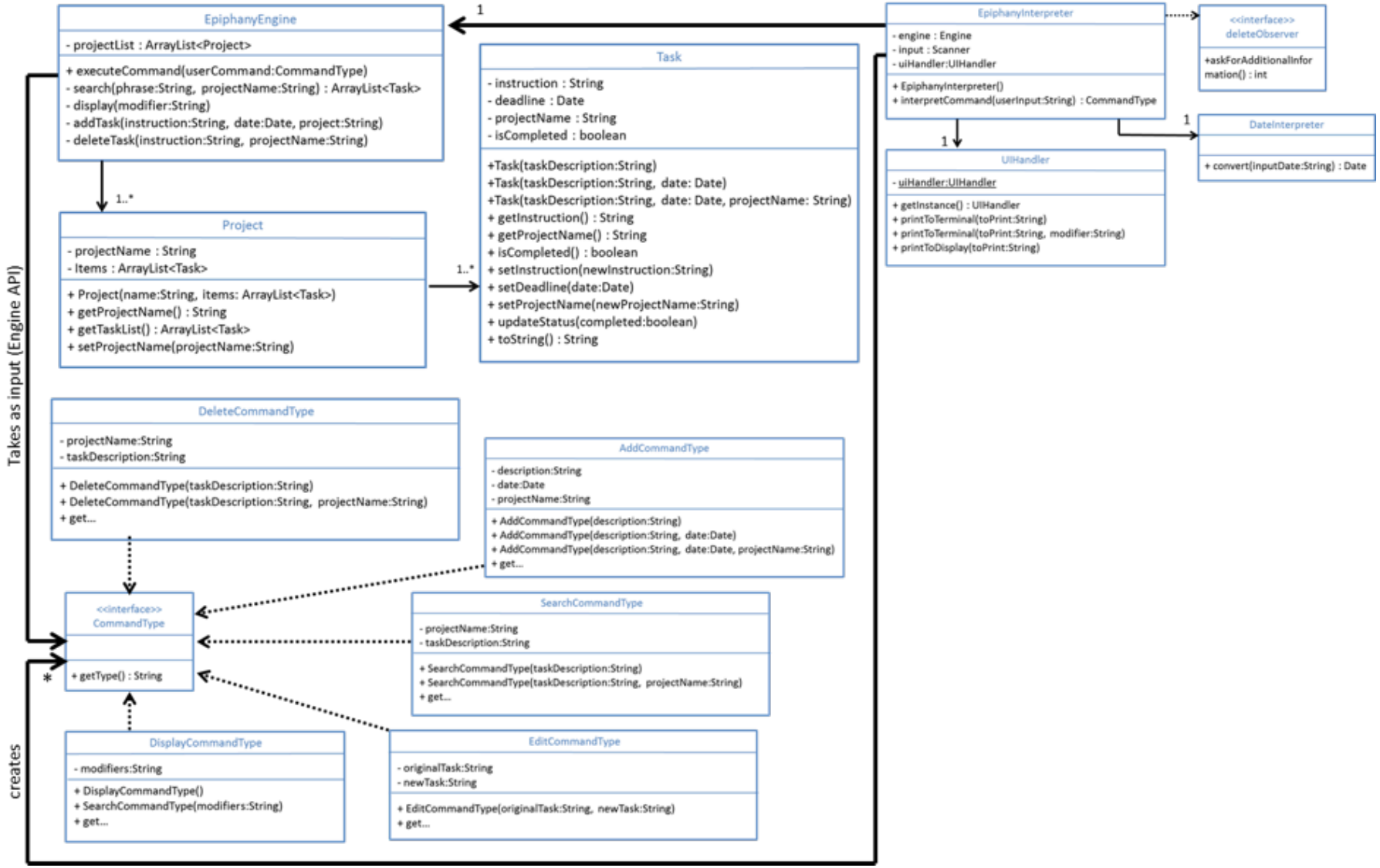
- *UI: The UI seen by users is written using the JavaFX library.*
- *Logic: The logic handles all the actual execution of the application.*
- *Storage: The storage component reads and writes to a single .txt file which acts as the database for the application.*

The diagram below shows how the code is organized into inside each component and dependencies among them. In this architecture Storage-UI-Logic represent an application of Model-View-Controller pattern



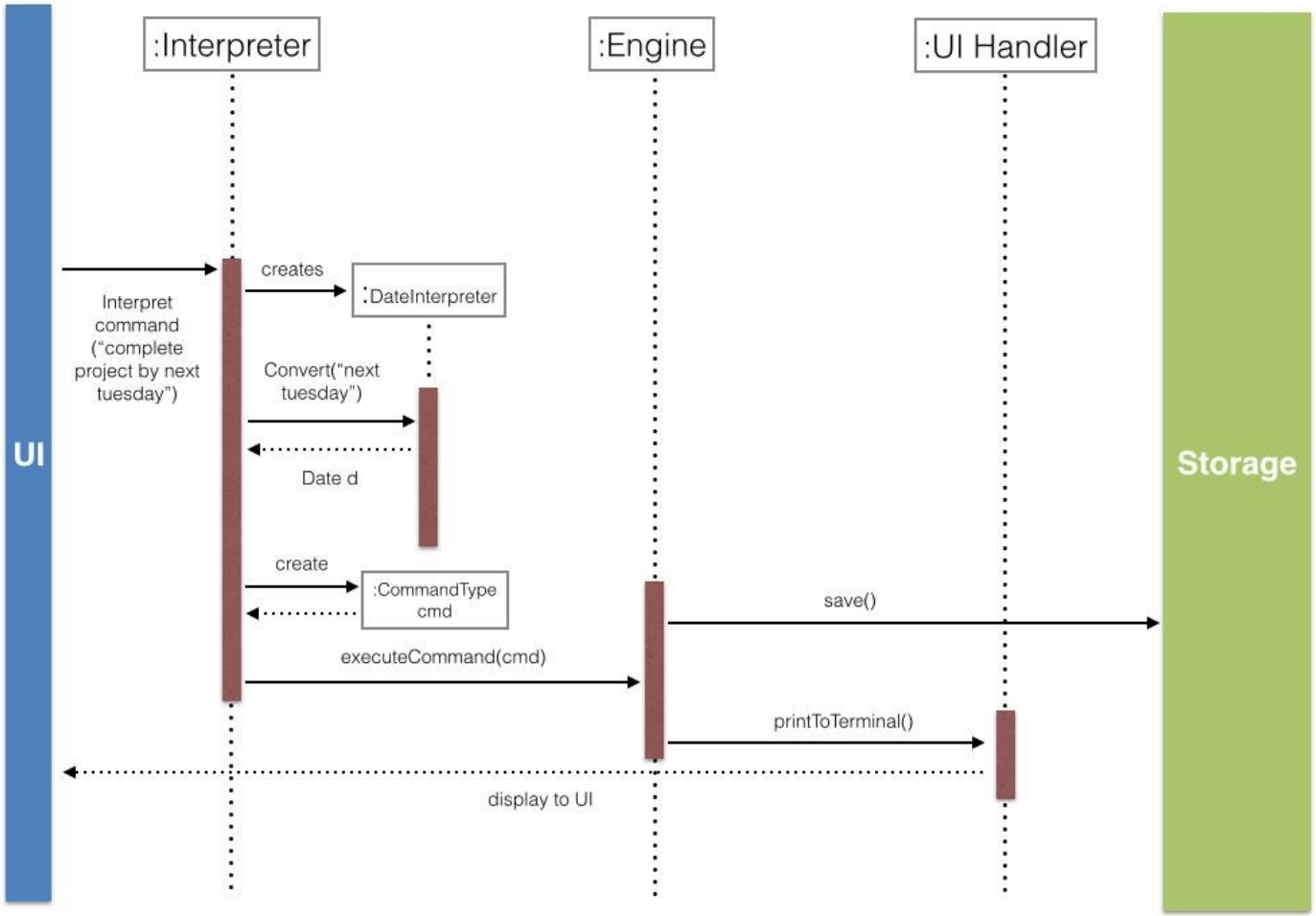
Logic

The diagram below shows the class diagram of the Logic Component.



Sequence Diagram

The following diagram demonstrates the program flow to add a task with a deadline.



Developer Guide

Application Programming Interface (API)

The following pages detail the API for the classes used in Epiphany.

EpiphanyInterpreter

This class parses the input from the user. The interpreter draws on several helper classes to accept simple and natural English, parses it and passes it onto the Engine Class via the CommandType Interface. This class follows the Observer pattern by implementing deleteObserver.

Engine

This class can be instantiated to perform all the operations that the program needs to perform. All the tasks would be stored in different projects of type project. Within these projects, we would have an ArrayList of Task which is used to store the instruction, the date as well as the name of the project that it belongs to. The interpreter would pass in the commands from the user and this class would be used to store, modify and update the projects that the user creates. It currently has the abilty to add, delete, search and display. Additionally, it has complete project management

Project

This class is used to create and modify projects that are created by the engine. A project contains a project name and an array list of Tasks. The tasks would be stored within the array list and the project name would be used to identify this project.

Task

This class aims to help in the creation and management of tasks. Each task contains an instruction, a deadline of class Date, a projectName which is the name of the project that the task is stored under and a boolean variable to check if the task has been completed.

CommandType

This is an interface which specifies the requirements of a command type, that would be created by the interpreter and passed to the engine for execution.

- **AddCommandType**
  This is a class that can be instantiated to represent an add command. It can support all types of add commands (with date and project, without date with project, with date without project and without date without project.) Each type of add command has a dedicated constructor.

- **EditCommandType**
  This is a class that can be instantiated to represent an edit command.

- **DeleteCommandType**
  This is a class that can be instantiated to represent a delete command. It can support all types of delete commands (all tasks, specific projects) Each type of delete command has a dedicated constructor.

- **DisplayCommandType**
  This is a class that can be instantiated to represent a display command. It can support all types of display commands (all tasks, specific projects) Each type of display command has a dedicated constructor.

- **SearchCommandType**
  This is a class that can be instantiated to represent an search command.

UIHandler

This is a singleton class that can be instantiated and used to perform all display to the user interface.

Logic

Class EpiphanyInterpreter

java.lang.Object
    Logic.EpiphanyInterpreter

All Implemented Interfaces:

`EpiphanyEngine.deleteObserver`

```
public class EpiphanyInterpreter
extends java.lang.Object
implements EpiphanyEngine.deleteObserver
```

This class parses the input from the user. The intepreter draws on several helper classes to accept simple and natural English, parses it and passes it onto the Engine Class via the CommandType Interface. This class follows the Observer pattern by implementing deleteObserver.

Author:

Amit and Abdulla

Constructor Summary

Constructors

Constructor and Description
`EpiphanyInterpreter()`

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`int`	`askForAdditionalInformation()`
`static void`	`main(java.lang.String[] args)` This is the main function which dictates the flow of the program.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

EpiphanyInterpreter

```
public EpiphanyInterpreter()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
                throws java.io.FileNotFoundException
```

This is the main function which dictates the flow of the program. All the functionality is abstracted out to other methods.

Parameters:

`args - which contains the file name (at index 0) entered by the user.`

Throws:

`java.io.FileNotFoundException`

askForAdditionalInformation

```
public int askForAdditionalInformation()
```

Specified by:

`askForAdditionalInformation in interface EpiphanyEngine.deleteObserver`

EpiphanyEngine

Class Engine

java.lang.Object
    EpiphanyEngine.Engine

```
public class Engine
extends java.lang.Object
```

This class can be instantiated to perform all the operations that the program needs to perform. All the tasks would be stored in different projects of type project. Within these projects, we would have an ArrayList of Task which is used to store the instruction, the date as well as the name of the project that it belongs to. The interpreter would pass in the commands from the user and this class would be used to store, modify and update the projects that the user creates. It currently has the abilty to add, delete, search and display. Additionally, it has complete project management. Missing: Storing projects in separate files (save).

Author:

Moazzam and Wei Yang

Field Summary

Fields

Modifier and Type	Field and Description
static java.util.ArrayList<EpiphanyEngine.Task>	**defaultProject**
static java.util.ArrayList<EpiphanyEngine.Project>	**EpiphanyMain**
static java.lang.String	**MESSAGE_ADD**
static java.lang.String	**MESSAGE_CLEAR**
static java.lang.String	**MESSAGE_CLEAR_EMPTY**
static java.lang.String	**MESSAGE_DELETE**
static java.lang.String	**MESSAGE_DELETE_INVALID**
static java.lang.String	**MESSAGE_DISPLAY**
static java.lang.String	**MESSAGE_DISPLAY_ERROR**
static java.lang.String	**MESSAGE_EXIT**
static java.lang.String	**MESSAGE_INVALID_SEARCH**
static java.lang.String	**MESSAGE_NO_ENTRY**
static java.lang.String	**MESSAGE_PROVIDE_ARGUMENT**
static java.lang.String	**MESSAGE_SORT**
static java.lang.String	**MESSAGE_SORTED**
static java.lang.String	**MESSAGE_WRONG_ENTRY**
static java.util.ArrayList<java.lang.String>	**projectNames**
static java.util.ArrayList<java.util.Date>	**testDate1**
static java.util.ArrayList<EpiphanyEngine.Task>	**testDateSort**

Constructor Summary

Constructors

Constructor and Description
Engine()

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
void	**deleteProject**(java.util.ArrayList<EpiphanyEngine.Task> projectName) Finds the project are removes it entirely.
void	**displayAll**() A function that helps to display all the projects including all the tasks stored within them.
void	**displayProject**(EpiphanyEngine.Project tempName) Displays all the tasks within any one of the projects.
void	**displayProjects**() Displays the names and indices of all the projects that exist.
void	**executeCommand**(CommandType userCommand) Interpreter passes in a command type object.
java.util.Date	**formatDate**(java.lang.String dateStr) This function helps to format the date.
void	**printToUser**(java.lang.String text, java.lang.String arg1, java.lang.String arg2)
java.lang.String	**removeFirstWord**(java.lang.String userCommand)
void	**sortDateInList**(java.util.ArrayList<java.util.Date> testDate) Sorts the date.
java.lang.String	**toString**(java.util.Date deadLine) Helps to format the date and return the formatted date.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

projectNames

```
public static java.util.ArrayList<java.lang.String> projectNames
```

defaultProject

```
public static java.util.ArrayList<EpiphanyEngine.Task> defaultProject
```

EpiphanyMain

```
public static java.util.ArrayList<EpiphanyEngine.Project> EpiphanyMain
```

testDate1

```
public static java.util.ArrayList<java.util.Date> testDate1
```

testDateSort

```
public static java.util.ArrayList<EpiphanyEngine.Task> testDateSort
```

MESSAGE_WRONG_ENTRY

```
public static final java.lang.String MESSAGE_WRONG_ENTRY
```

See Also:

Constant Field Values

MESSAGE_SORTED

```
public static final java.lang.String MESSAGE_SORTED
```

See Also:

Constant Field Values

MESSAGE_DELETE_INVALID

```
public static final java.lang.String MESSAGE_DELETE_INVALID
```

See Also:

Constant Field Values

MESSAGE_NO_ENTRY

```
public static final java.lang.String MESSAGE_NO_ENTRY
```

See Also:

Constant Field Values

MESSAGE_DELETE

```
public static final java.lang.String MESSAGE_DELETE
```

See Also:

Constant Field Values

MESSAGE_CLEAR_EMPTY

```
public static final java.lang.String MESSAGE_CLEAR_EMPTY
```

See Also:

Constant Field Values

MESSAGE_ADD

```
public static final java.lang.String MESSAGE_ADD
```

See Also:

Constant Field Values

MESSAGE_DISPLAY_ERROR

```
public static final java.lang.String MESSAGE_DISPLAY_ERROR
```

See Also:

Constant Field Values

MESSAGE_CLEAR

```
public static final java.lang.String MESSAGE_CLEAR
```

See Also:

Constant Field Values

MESSAGE_DISPLAY

```
public static final java.lang.String MESSAGE_DISPLAY
```

See Also:

Constant Field Values

MESSAGE_EXIT

```
public static final java.lang.String MESSAGE_EXIT
```

See Also:

Constant Field Values

MESSAGE_SORT

```
public static final java.lang.String MESSAGE_SORT
```

See Also:

Constant Field Values

MESSAGE_INVALID_SEARCH

```
public static final java.lang.String MESSAGE_INVALID_SEARCH
```

See Also:

Constant Field Values

MESSAGE_PROVIDE_ARGUMENT

```
public static final java.lang.String MESSAGE_PROVIDE_ARGUMENT
```

See Also:

Constant Field Values

Constructor Detail

Engine

```
public Engine()
```

Method Detail

displayAll

```
public void displayAll()
```

A function that helps to display all the projects including all the tasks stored within them.

displayProject

```
public void displayProject(EpiphanyEngine.Project tempName)
```

Displays all the tasks within any one of the projects.

Parameters:

`name - The name of the project that we wish to display.`

displayProjects

```
public void displayProjects()
```

Displays the names and indices of all the projects that exist.

deleteProject

```
public void deleteProject(java.util.ArrayList<EpiphanyEngine.Task> projectName)
```

Finds the project are removes it entirely.

Parameters:

`projectName - is the name of the project.`

formatDate

```
public java.util.Date formatDate(java.lang.String dateStr)
```

This function helps to format the date.

Parameters:

`dateStr - String that contains the date`

Returns:

`formatted date.`

sortDateInList

```
public void sortDateInList(java.util.ArrayList<java.util.Date> testDate)
```

Sorts the date.

Parameters:

`testDate - An ArryList of date that needs to be sorted.`

toString

```
public java.lang.String toString(java.util.Date deadLine)
```

Helps to format the date and return the formatted date.

Parameters:

`deadLine -`

Returns:

`the formatted date.`

executeCommand

```
public void executeCommand(CommandType userCommand)
```

Interpreter passes in a command type object. This method determines which type of command it is and uses the appropriate methods using the switch statements.

Parameters:

`userCommand - is the command that the interpreter send in.`

removeFirstWord

```
public java.lang.String removeFirstWord(java.lang.String userCommand)
```

printToUser

```
public void printToUser(java.lang.String text,
                        java.lang.String arg1,
                        java.lang.String arg2)
```

EpiphanyEngine

Class Project

java.lang.Object
    EpiphanyEngine.Project

```
public class Project
extends java.lang.Object
```

This class is used to create and modify projects that are created by the engine. A project contains a project name and an array list of Tasks. The tasks would be stored within the array list and the project name would be used to identify this project.

Author:

Moazzam and Wei Yang

Constructor Summary

Constructors

Constructor and Description
Project(java.lang.String name, java.util.ArrayList<EpiphanyEngine.Task> items)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
void	**createNewFile**(java.lang.String fileName, java.util.ArrayList<EpiphanyEngine.Task> items) Creates a new text file to store the new project file
java.lang.String	**getProjectName**()
java.util.ArrayList<EpiphanyEngine.Task>	**getTaskList**()
void	**setProjectName**(java.lang.String projectName)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Project

```
public Project(java.lang.String name,
               java.util.ArrayList<EpiphanyEngine.Task> items)
```

Method Detail

getProjectName

```
public java.lang.String getProjectName()
```

getTaskList

```
public java.util.ArrayList<EpiphanyEngine.Task> getTaskList()
```

setProjectName

```
public void setProjectName(java.lang.String projectName)
```

createNewFile

```
public void createNewFile(java.lang.String fileName,
                          java.util.ArrayList<EpiphanyEngine.Task> items)
                   throws java.io.IOException
```

Creates a new text file to store the new project file

Parameters:

`fileName - is the name of the file/project`

`items - is the ArrayList of items that is inside this project`

Throws:

`java.io.IOException`

EpiphanyEngine

Class Task

java.lang.Object
    EpiphanyEngine.Task

```
public class Task
extends java.lang.Object
```

This class aims to help in the creation and management of tasks. Each task contains an instruction, a deadline of class Date, a projectName which is the name of the project that the task is stored under and a boolean variable to check if the task has been completed.

Author:

Moazzam and Wei Yang

Constructor Summary

Constructors

Constructor and Description
Task(java.lang.String instruction)
Task(java.lang.String instruction, java.util.Date date)
Task(java.lang.String instruction, java.util.Date date, java.lang.String ProjectName) Overloaded constructors for the creation of tasks are shown below.
Task(java.lang.String instruction, java.lang.String projectName)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
java.lang.String	**getInstruction**()
java.lang.String	**getProjectName**()
boolean	**isCompleted**()
java.lang.String	**toString**()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Task

```
public Task(java.lang.String instruction,
            java.util.Date date,
            java.lang.String ProjectName)
```

Overloaded constructors for the creation of tasks are shown below. They differ in the type of arguments that they receive.

Parameters:

instruction - stores the actual task

date - stores the deadline

ProjectName - stores the name of the project that the task belongs to

Task

```
public Task(java.lang.String instruction)
```

Task

```
public Task(java.lang.String instruction,
            java.util.Date date)
```

Task

```
public Task(java.lang.String instruction,
            java.lang.String projectName)
```

Method Detail

getInstruction

```
public java.lang.String getInstruction()
```

getProjectName

```
public java.lang.String getProjectName()
```

isCompleted

```
public boolean isCompleted()
```

toString

```
public java.lang.String toString()
```

Overrides:

`toString in class java.lang.Object`

Logic.CommandType

Interface CommandType

`public interface` **`CommandType`**

This is an interface which specifies the requirements of a command type, that would be created by the interpreter and passed to the engine for execution.

Author:

`abdulla contractor and Amit Gamane`

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
java.lang.String	**getType**()

Method Detail

getType

```
java.lang.String getType()
```

Class AddCommandType

java.lang.Object
    Logic.CommandType.AddCommandType

All Implemented Interfaces:

`Logic.CommandType.CommandType`

```
public class AddCommandType
extends java.lang.Object
implements Logic.CommandType.CommandType
```

This is a class that can be instantiated to represent an add command. It can support all types of add commands (with date and project, without date with project, with date without project and without date without project.) Each type of add command has a dedicated constructor.

Author:

`abdulla contractor and amit gamane`

Constructor Summary

Constructors

Constructor and Description
AddCommandType(java.lang.String _description)
AddCommandType(java.lang.String _description, java.util.Date _date)
AddCommandType(java.lang.String _description, java.util.Date _date, java.lang.String _projectName)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
java.util.Date	**getDate**()
java.lang.String	**getDescription**()
java.lang.String	**getProjectName**()
java.lang.String	**getType**()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AddCommandType

```
public AddCommandType(java.lang.String _description)
```

AddCommandType

```
public AddCommandType(java.lang.String _description,
                      java.util.Date _date)
```

AddCommandType

```
public AddCommandType(java.lang.String _description,
```

```
                    java.util.Date _date,
                    java.lang.String _projectName)
```

Method Detail

getType

```
public java.lang.String getType()
```

Specified by:

`getType in interface Logic.CommandType.CommandType`

getDescription

```
public java.lang.String getDescription()
```

getDate

```
public java.util.Date getDate()
```

getProjectName

```
public java.lang.String getProjectName()
```

Logic.CommandType

Class EditCommandType

java.lang.Object
    Logic.CommandType.EditCommandType

```
public class EditCommandType
extends java.lang.Object
```

This is a class that can be instantiated to represent an edit command.

Author:

abdulla contractor and amit gamane

Constructor Summary

Constructors

Constructor and Description
EditCommandType(java.lang.String _originalTask, java.lang.String _newTask)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
java.lang.String	**getNewTask**()
java.lang.String	**getOriginalTask**()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

EditCommandType

```
public EditCommandType(java.lang.String _originalTask,
                       java.lang.String _newTask)
```

Method Detail

getOriginalTask

```
public java.lang.String getOriginalTask()
```

getNewTask

```
public java.lang.String getNewTask()
```

Logic.CommandType

Class DeleteCommandType

java.lang.Object
    Logic.CommandType.DeleteCommandType

All Implemented Interfaces:

`Logic.CommandType.CommandType`

```
public class DeleteCommandType
extends java.lang.Object
implements Logic.CommandType.CommandType
```

This is a class that can be instantiated to represent a delete command. It can support all types of delete commands (all tasks, specific projects) Each type of delete command has a dedicated constructor.

Author:

`abdulla contractor and amit gamane`

Constructor Summary

Constructors

Constructor and Description
DeleteCommandType(java.lang.String _taskDescription)
DeleteCommandType(java.lang.String _taskDescription, java.lang.String _projectName)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
java.lang.String	**getProjectName**()
java.lang.String	**getTaskDescription**()
java.lang.String	**getType**()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DeleteCommandType

```
public DeleteCommandType(java.lang.String _taskDescription)
```

DeleteCommandType

```
public DeleteCommandType(java.lang.String _taskDescription,
                         java.lang.String _projectName)
```

Method Detail

getType

```
public java.lang.String getType()
```

Specified by:

`getType in interface Logic.CommandType.CommandType`

getProjectName

```
public java.lang.String getProjectName()
```

getTaskDescription

```
public java.lang.String getTaskDescription()
```

Logic.CommandType

Class DisplayCommandType

java.lang.Object
    Logic.CommandType.DisplayCommandType

All Implemented Interfaces:

`Logic.CommandType.CommandType`

```
public class DisplayCommandType
extends java.lang.Object
implements Logic.CommandType.CommandType
```

This is a class that can be instantiated to represent a display command. It can support all types of display commands (all tasks, specific projects) Each type of display command has a dedicated constructor.

Author:

`abdulla contractor and amit gamane`

Constructor Summary

Constructors

Constructor and Description
DisplayCommandType()
DisplayCommandType(java.lang.String _modifiers)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
java.lang.String	**getModifiers**()
java.lang.String	**getType**()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DisplayCommandType

```
public DisplayCommandType()
```

DisplayCommandType

```
public DisplayCommandType(java.lang.String _modifiers)
```

Method Detail

getType

```
public java.lang.String getType()
```

Specified by:

getType in interface Logic.CommandType.CommandType

getModifiers

```
public java.lang.String getModifiers()
```

Logic.CommandType

Class SearchCommandType

java.lang.Object
    Logic.CommandType.SearchCommandType

All Implemented Interfaces:

`Logic.CommandType.CommandType`

```
public class SearchCommandType
extends java.lang.Object
implements Logic.CommandType.CommandType
```

This is a class that can be instantiated to represent an search command.

Author:

`abdulla contractor and amit gamane`

Constructor Summary

Constructors

Constructor and Description
SearchCommandType(java.lang.String _taskDescription)
SearchCommandType(java.lang.String _taskDescription, java.lang.String _projectName)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
java.lang.String	**getProjectName**()
java.lang.String	**getTaskDescription**()
java.lang.String	**getType**()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SearchCommandType

```
public SearchCommandType(java.lang.String _taskDescription)
```

SearchCommandType

```
public SearchCommandType(java.lang.String _taskDescription,
                         java.lang.String _projectName)
```

Method Detail

getType

```
public java.lang.String getType()
```

Specified by:

`getType in interface Logic.CommandType.CommandType`

getTaskDescription

```
public java.lang.String getTaskDescription()
```

getProjectName

```
public java.lang.String getProjectName()
```

Logic

Class UIHandler

java.lang.Object
    Logic.UIHandler

```
public class UIHandler
extends java.lang.Object
```

This is a singleton class that can be instantiated and used to perform all display to the user interface.

Author:

abdulla contractor and amit gamane

Method Summary

All Methods Static Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
static UIHandler	**getInstance**() Obtain a instance of the class
void	**printToDisplay**(java.lang.String toPrint) Prints paramter to Display Console
void	**printToTerminal**(java.lang.String toPrint) Prints parameter to Terminal.
void	**printToTerminal**(java.lang.String toPrint, java.lang.String modifier) Prints parameter inline to Terminal.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

getInstance

```
public static UIHandler getInstance()
```

Obtain a instance of the class

Returns:

UIHandler

printToTerminal

```
public void printToTerminal(java.lang.String toPrint)
```

Prints parameter to Terminal.

Parameters:

toPrint -

printToTerminal

```
public void printToTerminal(java.lang.String toPrint,
                            java.lang.String modifier)
```

Prints parameter inline to Terminal.

Parameters:

toPrint -

modifier -

printToDisplay

```
public void printToDisplay(java.lang.String toPrint)
```

Prints paramter to Display Console

Parameters:

toPrint -

Appendix A: User Stories. As a user…

[Likely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
addTask	add a task	can record tasks with a deadline.
addFloating	add a task without a deadline	can record tasks that I want to do some day.
deleteTask	delete a task	no longer have to track it.
completeTask	mark a task as completed	can view it in archive later.
ongoingTask	mark a task as ongoing	so that I can plan my work.
modifyTask	change details of a task	can have flexibility.
displayTasks	view all my tasks in one place.	can plan my work
readInput	add tasks through command line style arguments	can easily create new tasks
remindMe	recieve a reminder	remember to do things.
editTask	modify a task	can edit the task
undo	undo the last action (this does not stack multiple times)	can easily disregard a wrong command.

[Unlikely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
addProject	Create a project which can then contain tasks within it.	can organize my tasks into an intuitive format and make filtering easier.
powerSearch	Search through all my tasks, including date and notes field.	can easily search for anything i want
syncToGCal	Export my tasks to google calender	can easily integrate with any other calendar app i may have.
addNotes	add notes to any task	can elaborate on tasks without making the title messy

Appendix B: Product Survey

Product: iStudiez Pro **Documented by**: Amit Gamane

Strengths:

- Good organization
- Easy to navigate UI
- Ability to add/edit tasks with deadline
- Shortcuts for deadline(due next class etc.)
- Ability to add teacher information
- Integrates with iCal and shows how you schedule looks like
- Variable Reminder settings
- Holidays can be added

Weaknesses:

- Doesnt allow long term grade tracking, only task-based grading
- Backup works via email, could have included calendar syncing
- Displays all assignments due at once. I'd like to set an deadline to commit it to my schedule but may not want to see it on my list immediately.

Product: Todoist **Documented by:** Abdulla Contractor

Strengths:

- Simple and clean design
- Sends reminders by email
- The app is available on a wide variety of platforms, allowing you to access it from almost anywhere.
- You have the choice to collaborate with others on a project.
- You can create subtasks and by doing so break large tasks into smaller managable ones.
- You can quickly write due dates using normal language, such as "monday at 2pm".

Weaknesses:

- Does not sync to google calendar, it would be great if it could because the calendar on all my devices is integrated with google.

Product: http://todotxt.com/
Documented by: Tin Wei Yang
References: http://computers.tutsplus.com/tutorials/how-to-manage-your-tasks-with-todotxt--cms-20293
https://github.com/ginatrapani/todo.txt-android/releases/tag/release34

Strengths:

- Note taking at its purest, no reminders, checkboxes. Mimics the way how notes are taken down in real life - you write them down on a piece of paper
- Helps prioritize your to-do items & organize them into projects and contexts. This is done through the addition of a “+” tag followed by the project name at the end of the task. e.g. Do the dishes. +cleaning. Context can be added with an “@” sign followed by the name of the context. Priorities are designated with an uppercase, A-Z e.g. (A). A has a higher priority compared to B and so on.
- Open source project. Works in a variety of apps like Todour
- Command line editing of your task file
- Clean and minimal interface
- Modular - ability to combine with apps and the command line interface makes it as powerful as what the user would like.

Weaknesses:

- A slight learning curve initially.
- Unable to set time reminders, notifications
- Information is displayed in text, may not appeal to people who are more visual.

Product: Google Calendar integration with S planner **Documented by:** Moazzam Ali Khan

Strengths:

- Great synchronization features.
- Ability to synchronize with phone as well.
- Allows both tasking and calendar mode.
- Lots of options for calendar view; Daily, 4 days, weekly, monthly, etc.
- Great search function.
- Ability to combine with other apps.
- Email and sms reminders for flagged events.
- Allows easy addition of events, click from email.
- Daily Agenda mode; enables a daily agenda to be received via email at a set time.
- Allows sharing of calendar with others.

Weaknesses:

- Design is complicated. Needs getting used to.
- Too many settings which take users time to understand.