TE3201 Introduction to Software Engineering  
AY2021 Semester 2

**Project report**

**Name**: Lee Weihan Darren

**Student Number**: A0194169R

|  |
| --- |
| **User stories** Give some user stories (5-10) that match your project. Hint: refer to the textbook to find the meaning of ‘user stories’  [Epic] As a user, I want a tool to help me note down my tasks:   1. As a user, I want to be able to quickly add basic tasks so that I will remember what I need to do at different times of the day. 2. As a user, I want to be able to add more complex tasks with a deadline in the future so that I can keep track of what I have to do by a certain date 3. As a user, upon accomplishment of a certain task, I want to be able to mark it as done. 4. As a user, sometimes a task marked ‘done’ will have to be reopened again. I would have to have a way of changing the task status to undone. 5. As a user, I want to be given a choice to delete tasks so that my task list is streamlined.   [Epic] As a student, I have different needs for my task tool:   1. As a student, I can be on the move frequently. I want to my tasks saved when I shut down my computer, and yet view them again when I switch it on. 2. As a student, I want to be able to view my task progress at the end of the day so that I can track my performance. 3. As a student, I want to able to perform multiple operations on them so that I do not have to spend time doing repetitive actions. 4. As a student, I would like an interface to manage all tasks without delving into programming or the command line interface (CLI). |

|  |
| --- |
| **NFRs** Give some non-functional requirements (3-5) for your project.  Technical requirements:   1. Users should be able to add task list of arbitrary length. 2. Distinct error messages should be provided if the user enters in the wrong input.   Quality requirements:   1. Tool should be usable by a novice with no prior knowledge of how the tool would work. 2. Tool should be robust to unexpected input.   Product should be scalable and extensible by other developers if required. |
| **Welcome** Show the output T800 shows when you launch the program. |
| **Adding** Describe adding different types of task Note: ‘describe’ here (and in the subsequent sub-sections) means give examples of user commands and the app’s response for those commands. You may use screenshots.  Regular task:   * todo + TASK. (e.g. todo read book)     Deadline task:   * deadline + ‘by:’ + TASK. (e.g. deadline return book by: Tuesday) |
| **Listing** Describe listing tasks.  No specific command to list tasks because tasks are persistent in task pane.  As shown: |

|  |
| --- |
| **Marking** Describe marking/unmarking tasks as done.  Marking task 3 as done:   * ‘done’ + TASK NUMBER     Marking task 3 as undone:   * ‘pending’ + TASK NUMBER     What if user tries to mark a done task as done again or a pending task as pending?   * Mark pending task as pending      * Mark ‘done’ task as done. |

|  |
| --- |
| **Help** Describe how Monty provides a helpful instruction to users e.g., the result of a ‘help’ command  Helpful instruction panel is launched if user types ‘help’ explicitly. |
| **Saving** Give a sample of the tasks as they are stored in the hard disk.  Stored on harddisk (CSV):  T,read book,True  D,return book,False,Tuesday  T,watch movie,False |
| **ErrorHandling** Give different types of incorrect commands the app can handle and the corresponding error message given by the app.  Todo without a specific task (e.g. blank):    Deadline without a specific task (e.g. blank):    Deadline missing task:    Deadline missing deadline:    Deadline both task and deadline but with keyword ‘by:’:    Trying to mark non-existent tasks as pending or done:    Trying to delete, mark as done, or mark as pending non numerical input:    Trying to mass execute action using non-numerical inputs, or non-existent tasks.    Trying to execute mass action using invalid commands:    All other non-permitted commands not specified in ‘help’:    All commands are case-insensitive, e.g. ‘DeAdLine Read book By: 2pm” will be interpreted as a deadline item with task = ‘Read book’ and due date = ‘2pm’. |
| **Deleting** Describe deleting tasks.  Task list before:    Command entered:    Task list after: |
| **GUI**/**individual feature** If you implemented a GUI, give some screenshots. If you implemented an individual feature, describe that feature.  **Implemented GUI:**  Sample output with some tasks:    **MassOps function:**  Allows mass deletion, mark pending or done of all specified tasks. Items are separated with a whitespace, e.g. ‘mass delete 3 5’ for deletion of tasks 3 and 5. Accepts non-numerical input as it will interpret only the numerical parts.  E.g. ‘mass done 1 e3 app1e 2e 3’ will be interpreted as executing ‘done’ on items 1 and 3 of the local task list.  Task list before:    Command: ‘mass delete 3 5’    Task list after:    **Search function:**  Searches all tasks for keywords matching search string input. Returns index of item containing specified string.  Search is case insensitive, e.g. ‘apple’, ‘ApPlE’ and ‘aPpLe’ at different indexes will be interpreted differently, however only full search string with whitespaces at each end will be considered. E.g., ‘applepie’ will not be considered as matching the keyword string.  E.g  Task list before:    Command: ‘find apple’:    **Wipe function:**  Clears all tasks from task list. Useful for debugging.  E.g  Task list before:    Command: ‘wipe’    Task list after: |
| **Other features** Describe other features you implement (i.e., not described above), if any e.g., optional increments.  **Progress:**  Users can view their current task progress in the session, i.e. how many todo, or deadline tasks have been done in the current session. Task progress counter will be reset upon exit or ‘wipe’.  Sample:  Before marking any tasks as done:    Marking tasks 3, 4, 5 as done:    Asking for progress:    Final output:    **Arbitrary input requirements:**  Users allowed to input tasks or deadlines of ANY length and have it properly displayed.  Sample:    Benefit: Users can enter tasks of arbitrary length and view them properly, instead of the 14 character and 8-character limit of the ‘description’ and ‘deadline’ respectively.  This is more crucial for the GUI because the limits of the GUI display would improperly print the tasks if longer than the lengths specified above. |

|  |
| --- |
| **OOP/UML** Give a class diagram to match your code. Include examples of (if applicable) in the diagram: classes, some attributes/methods, associations, inheritance, navigability, association labels and roles, multiplicity, class-level members    Give at least one object diagram illustrating the state of your program when the user has added at least 2 tasks.  2 tasks added:   * Deadline task: “Return Book by: Tuesday” * ToDo task: “Read Book” |

|  |
| --- |
| **UnitTests**: Give the code of 2-3 unit tests (if any) from your code.  Testing add\_item() and add\_deadline\_item() methods in TaskManager.py. |
| **Suggested test commands** Give a list of commands a tester can execute in sequence to examine your product. Cover all features in a reasonable order. E.g,   1. Help 2. Todo borrow book 3. Todo read book 4. Deadline return book by: Sunday 5. Done 1 6. Progress 7. Mass done 2 3 8. Progress 9. find read 10. Pending 3 11. Progress 12. Delete 1 13. Wipe 14. Exit |

# Appendix: Code

Give all your code here. Indicate filename too. Some examples given below.

**Main.py**

1. import datetime
2. from tkinter import Tk, Entry, Text, LEFT, RIGHT, END
3. import itertools
4. import sys
5. import deadline as dl
6. import taskmanager as tm
7. import userinterface as ui
9. class GUI:
11. def \_\_init\_\_(self, task\_manager):
13. # create ui class instance
14. self.ui\_object = ui.UserInterface()
16. self.task\_manager = task\_manager
17. self.window = Tk()
18. self.window.geometry('800x700') # set Window size
19. self.window.title('Task Terminator T800') # set Window title
21. self.input\_box = Entry(self.window) # create an input box
22. self.input\_box.pack(padx=5, pady=5, fill='x') # make the input box fill the width of the Window
23. self.input\_box.bind('<Return>', self.command\_entered) # bind the command\_entered function to the Enter key
24. self.input\_box.focus() # set focus to the input box
26. # add a text area to show the chat history
27. self.history\_area = Text(self.window, width="50")
28. self.history\_area.pack(padx=5, pady=5, side=LEFT, fill="y")
29. self.output\_font = ('Courier New', 11)
30. self.history\_area.tag\_configure('error\_format', foreground='red', font=self.output\_font)
31. self.history\_area.tag\_configure('success\_format', foreground='green', font=self.output\_font)
32. self.history\_area.tag\_configure('normal\_format', font=self.output\_font)
34. # add a text area to show the list of tasks
35. self.list\_area = Text(self.window)
36. self.list\_area.pack(padx=5, pady=5, side=RIGHT, fill="both")
37. self.list\_area.tag\_configure('normal\_format', font=self.output\_font)
38. self.list\_area.tag\_configure('pending\_format', foreground='red', font=self.output\_font)
39. self.list\_area.tag\_configure('done\_format', foreground='green', font=self.output\_font)
41. # show the welcome message and the list of tasks
42. self.update\_chat\_history('start', 'Welcome to T800!', 'success\_format')
43. self.update\_task\_list(self.task\_manager.items, self.ui\_object)
45. def update\_chat\_history(self, command, response, status\_format):
46. """
47. status\_format: indicates which color to use for the status message
48. can be 'error\_format', 'success\_format', or 'normal\_format'
49. """
50. current\_time = datetime.datetime.now().strftime("%H:%M:%S")
51. self.history\_area.insert(1.0, '-' \* 40 + '\n', 'normal\_format')
52. self.history\_area.insert(1.0, '>>> ' + str(response) + '\n', status\_format)
53. self.history\_area.insert(1.0, 'You said: ' + str(command) + '\n', 'normal\_format')
54. self.history\_area.insert(1.0, current\_time + '\n', 'normal\_format')
56. def update\_task\_list(self, tasks, ui\_object):
58. self.list\_area.delete('1.0', END) # clear the list area
59. self.list\_area.insert(END, self.ui\_object.show\_greeting())
60. if len(tasks) == 0:
61. self.list\_area.insert(END, '>>> Nothing to list', 'normal\_format')
62. else:
63. self.\_\_overflow\_print(tasks)
65. def \_\_overflow\_print(self, tasks):
66. for i, task in enumerate(tasks):
67. output\_format = 'done\_format' if task.is\_done else 'pending\_format'
69. deadline\_arr = []
70. desc\_arr = []
72. if isinstance(task, dl.Deadline):
73. to\_print = str(task)[:6] + '|' + str(i+1).center(6) + '| ' + str(task)[6:20] + \
74. ' | ' + str(task.by)[:8] + '\n'
75. self.list\_area.insert(END, to\_print, output\_format) # print first 8 chars of 'deadline' only
77. if len(str(task.by)) > 8:
78. deadline\_arr = self.\_\_string\_splitter(deadline\_arr, task.by, 8) [1:] # if longer than 8 char, split string
79. else:
80. to\_print = str(task)[:6] + '|' + str(i+1).center(6) + '| ' + str(task)[6:20] + \
81. ' | ' + '-' + '\n'
82. self.list\_area.insert(END, to\_print, output\_format) # print first 14 chars of 'deadline' only
84. if len(str(task.description)) > 14:
85. desc\_arr = self.\_\_string\_splitter(desc\_arr, task.description, 14)[1:] # if longer than 14 char, split string
87. for combination in itertools.zip\_longest(desc\_arr, deadline\_arr, fillvalue=""):
88. to\_print = " "\*15 + combination[0].ljust(14) + " "\*3 + combination[1].ljust(8) + '\n'
89. self.list\_area.insert(END, to\_print, output\_format) # iterate thru 2 lists simultanously, filling up diff with ""
90. self.list\_area.insert(END, """----------------------------------------------\n""")

93. def \_\_string\_splitter(self, arr, string, split\_length):
94. """ Recursively splits the string greedily in length specified by split\_length"""
95. if len(string) < split\_length:
96. arr.append(string)
97. return arr
98. else:
99. arr.append(string[:split\_length])
100. return self.\_\_string\_splitter(arr, string[split\_length:], split\_length)
102. def clear\_input\_box(self):
103. self.input\_box.delete(0, END)
105. def command\_entered(self, event):
106. command = None
107. try:
108. command = self.input\_box.get().strip()
109. if command.lower() == 'exit':
110. self.window.destroy()
111. sys.exit()
112. output = self.task\_manager.execute\_command(command)
113. self.update\_chat\_history(command, output, 'success\_format')
114. self.update\_task\_list(self.task\_manager.items, self.ui\_object)
115. self.clear\_input\_box()
116. self.task\_manager.save\_data()
117. except Exception as e:
118. self.update\_chat\_history(command, str(e), 'error\_format')
120. def start(self):
121. self.window.mainloop()
123. if \_\_name\_\_ == '\_\_main\_\_' :
124. GUI(tm.TaskManager()).start()

**Taskmanager.py**

1. import re
2. from todo import ToDo as td
3. from deadline import Deadline as dl
4. from storagemanager import StorageManager as sm
6. import exceptions as ex
7. import userinterface as ui
9. FILENAME = 'monty7.csv'
11. class TaskManager:
12. """
13. TaskManager class will execute task-related commands
14. """
15. FILENAME = 'monty7.csv'
16. items = []
18. def \_\_init\_\_(self):
19. self.storage = sm(FILENAME)
20. self.storage.load\_data(self.items)
22. def save\_data(self):
23. self.storage.save\_data(self.items)
25. def add\_item(self, user\_input):
26. """
27. Adds a 'ToDo' type item into the local list.
29. Parameters
30. ----------
31. user\_input:
32. All input after keyword 'todo' is considered valid.
34. Raises
35. ------
36. IndexError
37. If user provides blank input
39. Returns
40. -------
41. Information on task added.
43. """
44. command\_parts = user\_input.strip().split(' ', 1)
45. try:
46. self.items.append(td(command\_parts[1], False))
47. return ("New item: " + "'" + command\_parts[1] + "'" + " added")
48. except IndexError as ie:
49. raise IndexError("INPUT: todo \"task\"") from ie

52. def add\_deadline\_item(self, user\_input):
53. """
54. Adds a 'Deadline' type item into the list.
56. Parameters
57. ----------
58. user\_input : Requires a keyword 'by:'
59. All input after keyword 'deadline' is considered valid.
61. Raises
62. ------
63. ex.InvalidDeadlineInput:
64. When 'by' keyword is not present in input (case insensitive)
66. ex.NoTaskError:
67. When no task is provided but 'by:' keyword is present
69. ex.NoDueDateError:
70. When no due date is provided but 'by:' keyword is present
72. ex.NoDueNoTaskError:
73. When both task & due date is not provided but keyword is present
75. Returns
76. -------
77. Information on task added.
79. """
80. if re.search(" by:", user\_input, re.IGNORECASE):
81. command\_parts = user\_input.strip().split(' ', 1)
82. arr = re.split("by:", command\_parts[1], 1, re.IGNORECASE)
83. (task, due) = [x.strip() for x in arr]
85. if not due and not task:
86. raise ex.NoDueNoTaskError("INPUT: deadline \"task\" by: \"due date\"")
87. elif not due:
88. raise ex.NoDueDateError("No due date provided! INPUT: deadline \"task\" by: \"due date\"" )
89. elif not task:
90. raise ex.NoTaskError("Provide a task! INPUT: deadline \"task\" by: \"due date\"")
92. self.items.append(dl(task, False, due))
93. return ("New item: " + "'" + task + "'" + " added. " + "Deadline: " + "'" + due + "'")
94. else:
95. raise ex.InvalidDeadlineInput("Missing 'by' keyword! INPUT: deadline \"task\" by: \"due date\"")
97. def mark\_item\_as\_done(self, user\_input):
98. """
99. Marks a specified item as 'done'.
101. Parameters
102. ----------
103. user\_input :
104. A number within bounds of the current tasklist.
106. Returns
107. -------
108. Information on task marked 'done' or if task has been mark 'done' prior.
110. """
112. index\_as\_string = user\_input[5:].strip()
113. index\_to\_remove = self.\_\_index\_check(index\_as\_string)
114. for i, obj in enumerate(self.items):
115. if i == index\_to\_remove:
116. if obj.is\_done:
117. return ("Item: " + "'" + obj.description + "'" + " has been done already")
118. obj.mark\_as\_done()
119. return ("Item: " + "'" + obj.description + "'" + " marked as done")
121. def mark\_item\_as\_pending(self, user\_input):
122. """
123. Marks a specified item as 'pending'.
125. Parameters
126. ----------
127. user\_input :
128. A number within bounds of the current tasklist.
130. Returns
131. -------
132. Information on task marked 'pending' or if task has been mark 'pending' prior.
134. """
135. index\_as\_string = user\_input[8:].strip()
136. index\_to\_remove = self.\_\_index\_check(index\_as\_string)
137. for i, obj in enumerate(self.items):
138. if i == index\_to\_remove:
139. if not obj.is\_done:
140. return ("Item: " + "'" + obj.description + "'" + " is already pending")
141. obj.mark\_as\_pending()
142. return ("Item: " + "'" + obj.description + "'" + " marked as pending")
144. def \_\_index\_check(self, string):
145. """
146. Helper function to check for valid input for mark\_as\_done and
147. mark\_as\_pending methods.
149. Parameters
150. ----------
151. string :
152. A single number that was obtained from input of the mark\_as\_done
153. or mark\_as\_pending functions
155. Raises
156. ------
157. ValueError
158. Non-numerical input.
159. ex.ZeroInputError:
160. Invalid element '0' specified
161. IndexError
162. No element at specified index.
164. Returns
165. -------
166. Provided number - 1
168. """
169. try:
170. index = int(string.strip())
171. except ValueError as ve:
172. raise ValueError('"{}" is not a number'.format(string)) from ve
173. if index < 1:
174. raise ex.ZeroInputError('Index must be greater than 0')
175. try:
176. if self.items[index - 1]:
177. return index - 1
178. except IndexError as ie:
179. raise IndexError('No item at index: {}'.format(string)) from ie

182. def delete\_item(self, user\_input):
183. """
184. Deletes item from local tasklist.
186. Parameters
187. ----------
188. user\_input :
189. A number within bounds of the current tasklist.
191. Raises
192. ------
193. IndexError
194. No element at specified index.
195. ValueError
196. Non-numerical input.
198. Returns
199. -------
200. Information on task deleted.
202. """
203. try:
204. delete\_string = int(user\_input[7:].strip())
205. if delete\_string == 0:
206. raise IndexError
207. deleted\_item = self.items.pop(delete\_string-1)
208. return ("Task: " + "'" + deleted\_item.description + "'" + " deleted from the list")
209. except IndexError as ie:
210. raise IndexError("There is no list item at the number you typed!") from ie
211. except ValueError as ve:
212. raise ValueError("Only integers accepted as input") from ve
214. def get\_current\_progress(self):
215. """
216. Obtains progress of current session, ie.
217. how many ToDos and Deadlines tasks marked as done.
218. Tasks marked pending after being marked done will be
219. taken into account.
221. Returns
222. -------
223. Progress for this session.
225. """
226. status = {'Todo': 0, 'Deadline': 0}
227. status['Todo'] = td.progress\_check()
228. status['Deadline'] = dl.progress\_check()
229. return("""Progress for this session:
230. | ToDos: {} | Deadlines: {} |""".format(status['Todo'], status['Deadline']))
232. def mass(self, user\_input):
233. """
234. Executes mass execution of delete, pending or done sub methods.
236. Parameters
237. ----------
238. user\_input :
239. Requires either 'delete', 'done', or 'pending' to be specified.
241. Raises
242. ------
243. ex.InvalidMassInputError
244. When action keywords 'delete', 'pending', or 'done' is not specified
246. Returns
247. -------
248. A internal call to private method mass\_execute.
250. """
252. command = user\_input[5:].strip()
253. if command.startswith('delete'):
254. return self.\_\_mass\_execute(command.lower(), 7)
255. elif command.startswith('done'):
256. return self.\_\_mass\_execute(command.lower(), 5)
257. elif command.startswith('pending'):
258. return self.\_\_mass\_execute(command.lower(), 8)
259. else:
260. raise ex.InvalidMassInputError("INPUT: 'mass' + delete/done/pending + args\*\*")
262. def \_\_mass\_execute(self, command, strip\_len):
263. """
264. Executes 'command' determined by strip\_len, sequentially.
265. Will execute all numbers regardless of whether input contains non-numerical
266. input or not.
268. E.g. 'mass done 1 e3 app0e 2' results in task 1 and 2 marked as done.
270. Parameters
271. ----------
272. command : Number
273. Numbers to execute operations on
274. function\_name : method
275. String representing method to execute
276. strip\_len : Number
277. Length of string to strip()
279. Raises
280. ------
281. ex.InvalidMassInputError
282. if no numbers are supplied BUT keyword is present, e.g.
283. 'mass delete abce1de'
285. Returns
286. -------
287. Information on task and what was executed on.
289. """
290. string = command[strip\_len:]
291. back\_string = command[:strip\_len]
292. str\_arr = [int(s) for s in string.split() if s.isdigit()]
293. str\_arr.sort(reverse=True)
294. for i in str\_arr:
295. self.execute\_command(back\_string + str(i))
296. if str\_arr:
297. return ("Executed mass action '{}' on items {}!".format(back\_string.strip(), str\_arr))
298. else:
299. raise ex.InvalidMassInputError("Nothing was done! Check your input.")
301. def find(self, user\_input):
302. """
303. Search list of tasks for task matching user input. Search is
304. case-insensitive!
306. Parameters
307. ----------
308. user\_input : STRING
309. User desired string to search for
311. Returns
312. -------
313. STRING
314. String with item(s) location or string informing of failure to find item.
316. """
318. item\_locations = []
319. command = user\_input[5:].strip()
320. pattern = "(?:^|\W)" + str(command) + "(?:$|\W)"
322. for index, item in enumerate(self.items):
323. if re.search(pattern, item.description, re.IGNORECASE):
324. item\_locations.append(index+1)
325. else:
326. continue
328. if not item\_locations:
329. raise Exception("Item not found in tasklist!")
330. else:
331. return ("Item(s) '{}' found at index {}". format(str(command.lower()), str(item\_locations)))
333. def clear\_screen(self):
334. """
335. Wipes screen of all tasks
337. """
338. self.items = []
339. return ("All tasks removed from tasklist!")
341. def execute\_command(self, command):
342. """
343. Main function to execute commands. All commmands are case-insensitive,
344. i.e. 'DeAdlIne read book BY: 2pm' will add a Deadline item into list
345. with a deadline of 2pm.
347. Parameters
348. ----------
349. command : String
350. User input from GUI box using Tkinter
352. Raises
353. ------
354. Exception
355. If command entered is not valid.
357. Returns
358. -------
359. A call to associated command to execute with keyword parameters.
361. """
362. string = command.lower()
364. if string.startswith('help'):
365. return ui.UserInterface.get\_help()
366. elif string.startswith('progress'):
367. return self.get\_current\_progress()
368. elif string.startswith('wipe'):
369. return self.clear\_screen()
370. elif string.startswith('done'):
371. return self.mark\_item\_as\_done(command)
372. elif string.startswith('pending'):
373. return self.mark\_item\_as\_pending(command)
374. elif string.startswith('delete'):
375. return self.delete\_item(command)
376. elif string.startswith('todo'):
377. return self.add\_item(command)
378. elif string.startswith('deadline'):
379. return self.add\_deadline\_item(command)
380. elif string.startswith('find'):
381. return self.find(command)
382. elif string.startswith('mass'):
383. return self.mass(command)
384. else:
385. raise Exception('Command not recognized. Input \'help\' to see all available commands.')

**StorageManager.py**

1. # -\*- coding: utf-8 -\*-
2. import csv
3. import deadline as dl
4. import todo as td
6. class StorageManager:
8. def \_\_init\_\_(self, filename):
9. self.filename = filename
11. def load\_data(self, items):
12. """
13. Loads data from csv specified.
15. If csv does not exist, a new CSV file is created
16. in the same directory as main.
18. Items are loaded from CSV into the item list.
20. Returns
21. -------
22. None.
24. """
25. self.\_\_create\_file\_if\_missing(self.filename)
26. with open(self.filename, 'r') as csvfile:
27. file\_handler = csv.reader(csvfile)
28. for row in file\_handler:
29. if not row:
30. continue
31. self.\_\_load\_item\_from\_csv\_line(row, items)
32. return
34. def \_\_create\_file\_if\_missing(self, filename):
35. """
36. Creates a file.
38. If the named csv file is locked for editing, a permission error is
39. raised to console.
41. Returns
42. -------
43. None.
45. """
46. try:
47. open(filename, 'a').close()
48. except PermissionError as pe:
49. raise PermissionError("Error creating file, check permissions.") from pe

52. def \_\_load\_item\_from\_csv\_line(self, row, items):
53. """
54. From the specified CSV, it will read the CSV row by row and create todo
55. and deadline objects respectively.
57. Parameters
58. ----------
59. row : Each line in CSV passed from csv\_reader
60. If CSV line starts with 'T', create ToDo instance, else
61. if line starts with 'D', create Deadline instance.
63. Raises
64. ------
65. IndexError
66. Raises IndexError if unable to get indexes of row.
68. Returns
69. -------
70. None.
72. """
73. try:
74. if row[0] == 'T':
75. items.append(td.ToDo(row[1], True if row[2] == 'True' else False))
76. elif row[0] == 'D':
77. items.append(dl.Deadline(row[1], True if row[2] == 'True' else False, row[3]))
78. except IndexError:
79. raise IndexError
80. return
82. def save\_data(self, items):
83. """
84. Method to save data to external CSV specified in attribute.
86. Returns
87. -------
88. None.
90. """
91. with open(self.filename, "w", newline='') as csvfile:
92. output = csv.writer(csvfile)
93. for item in items:
94. if isinstance(item, dl.Deadline):
95. output\_to\_file = ["D",item.description,item.is\_done,item.by]
96. else:
97. output\_to\_file = ["T",item.description,item.is\_done]
98. output.writerow(output\_to\_file)

**Todo.py**

1. class ToDo:
3. \_progress = 0 # class-level variable
4. TYPE\_KEY = 'T'
6. def \_\_init\_\_(self, description, status):
7. self.description = description
8. self.is\_done = status
10. def \_\_str\_\_(self):
11. return self.\_\_status\_as\_icon().center(6) + self.description.ljust(14)
13. def mark\_as\_done(self):
14. if not self.is\_done: # increment progress if needed
15. ToDo.\_progress = ToDo.\_progress + 1
16. self.is\_done = True
18. def mark\_as\_pending(self):
19. if self.is\_done: # decrement progress if needed
20. if ToDo.\_progress != 0:
21. ToDo.\_progress = ToDo.\_progress - 1
22. self.is\_done = False
24. def \_\_status\_as\_icon(self):
25. return 'X' if self.is\_done else '-'
27. def as\_csv(self):
28. """ Return the details of todo object as a list,
29. suitable to be stored in a csv file.
30. """
31. return [self.TYPE\_KEY, self.description, 'done' if self.is\_done else 'pending']
33. @classmethod
34. def progress\_check(cls):
35. return cls.\_progress

**Deadline.py**

1. import todo as td
3. class Deadline (td.ToDo):
5. \_progress = 0 # class-level variable
6. TYPE\_KEY = 'D'
8. def \_\_init\_\_(self, description, status, by):
9. super().\_\_init\_\_(description, status)
10. self.by = by
12. def \_\_str\_\_(self):
13. s = super().\_\_str\_\_()
14. return s[:-1] + " " + self.by
16. def as\_csv(self):
17. c = super().as\_csv()
18. c.append(self.by)
19. return c
21. def mark\_as\_done(self):
22. if not self.is\_done: # increment progress if needed
23. Deadline.\_progress = Deadline.\_progress + 1
24. self.is\_done = True
26. def mark\_as\_pending(self):
27. if self.is\_done: # decrement progress if needed
28. if Deadline.\_progress != 0:
29. Deadline.\_progress = Deadline.\_progress - 1
30. self.is\_done = False

**Exceptions.py**

1. class Error(Exception):
2. """Base class for other exceptions"""
3. pass
5. class ZeroInputError(Error):
6. """Raised when the input is 0"""
7. pass
9. class InvalidMassInputError(Error):
10. """Raised when nothing was done in a mass action command"""
11. pass
13. class InvalidDeadlineInput(Error):
14. """Raised when 'by:' keyword is not entered. Also base class for other
15. deadline exceptions'"""
16. pass
18. class NoDueDateError(InvalidDeadlineInput):
19. """Raised when the ONLY due date is blank"""
20. pass
22. class NoDueNoTaskError(InvalidDeadlineInput):
23. """Raised when the both due date AND task is blank"""
24. pass
26. class NoTaskError(InvalidDeadlineInput):
27. """Raised when the both due date AND task is blank"""
28. pass

**Testcase.py**

1. import unittest
2. import exceptions as ex
3. import taskmanager as tm
5. # test class
6. class TestSearch(unittest.TestCase):
8. test\_task\_manager = tm.TaskManager()
10. def test\_add\_item(self):
11. with self.assertRaises(IndexError):
12. self.test\_task\_manager.add\_item("todo")
13. with self.assertRaises(IndexError):
14. self.test\_task\_manager.add\_item("todo ")
15. self.assertEqual(self.test\_task\_manager.add\_item("todo foo bar"), "New item: 'foo bar' added")
16. self.assertEqual(self.test\_task\_manager.add\_item("todo 1234567"), "New item: '1234567' added")
17. self.assertEqual(self.test\_task\_manager.add\_item("todo !@#$%^&\*"), "New item: '!@#$%^&\*' added")
18. self.assertEqual(self.test\_task\_manager.add\_item("todo todo"), "New item: 'todo' added")
20. def test\_add\_deadline\_item(self):
21. self.assertEqual(self.test\_task\_manager.add\_deadline\_item("deadline foo by: bar")
22. , "New item: 'foo' added. Deadline: 'bar'")
23. self.assertEqual(self.test\_task\_manager.add\_deadline\_item("deadline 123 by: 567")
24. , "New item: '123' added. Deadline: '567'")
25. self.assertEqual(self.test\_task\_manager.add\_deadline\_item("deadline deadline by: deadline")
26. , "New item: 'deadline' added. Deadline: 'deadline'")
27. self.assertEqual(self.test\_task\_manager.add\_deadline\_item("deadline !@#$% by: ((\*&^)) ")
28. , "New item: '!@#$%' added. Deadline: '((\*&^))'")

31. with self.assertRaises(ex.InvalidDeadlineInput):
32. self.test\_task\_manager.add\_deadline\_item("deadline")
33. with self.assertRaises(ex.NoTaskError):
34. self.test\_task\_manager.add\_deadline\_item("deadline by: foo")
35. with self.assertRaises(ex.NoDueDateError):
36. self.test\_task\_manager.add\_deadline\_item("deadline foo by: ")
37. with self.assertRaises(ex.NoDueNoTaskError):
38. self.test\_task\_manager.add\_deadline\_item("deadline by: ")
39. with self.assertRaises(ex.NoTaskError):
40. self.test\_task\_manager.add\_deadline\_item("deadline by: by:")
41. with self.assertRaises(ex.InvalidDeadlineInput):
42. self.test\_task\_manager.add\_deadline\_item("deadline foo by")
44. # activate the test runner
45. if \_\_name\_\_ == '\_\_main\_\_':
46. unittest.main()