Deakin University

Object Oriented Development

OnTrack Submission

BuggySoft: Program Design and Class Composition

Submitted By:
Peter Stacey
pstacey
2020/04/25 19:07

Tutor:
Dipto Pratyaksa

Outcome	Weight
Evaluate Code	♦♦♦♦♦
Principles	♦♦♦♦◊
Build Programs	♦♦♦♦◊
Design	♦♦♦♦◊
Justify	♦♦♦◊◊

Taking an existing codebase and being able to reason it, directly related to outcome 1, and this task also extends beyond that by requiring us to maintain and refactor the code to a more robust solution to the original problem. This goes straight to outcomes 3 and 4 and then the final part of the task, requiring the addition of functionality, directly relates even further, to outcomes 2, 3 and 4. Together with the submitted code and my video, the evidence aligns with outcome 5.

April 25, 2020

```
using System;
using System.Linq;
using System.Collections.Generic;

namespace DuplicateCode
{
    class RevisedCode
    {
        /// <summary>
        /// Outputs a prompt for input and returns the input string
        /// <summary>
        /// <returns>
        /// The input of the user as a string
        /// </summary>
        public static string ReadString(string prompt)
        {
            Console.WriteLine(prompt);
            Console.Write(">> ");
            return Console.ReadLine();
        }

        /// <summary>
        /// Returns the length of the longest list in the dictionary
        /// </summary>
        /// <returns>
        /// Length of the longest list as an integer
        /// </returns>
        /// <exception cref="System.ArgumentNullException">Thrown if the
        /// dictionary or any list is null
        /// </exception>
        static int MaximumLength(Dictionary<string, List<string>> tasks)
        {
            return tasks.Values.Max(list => list.Count);
        }

        /// <summary>
        /// Prints the current list of tasks by category, out to the
        /// console
        /// </summary>
        static void PrintTasks(Dictionary<string, List<string>> tasks)
        {
            int max = MaximumLength(tasks);
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.WriteLine(new string(' ', 12) + "CATEGORIES");
            Console.WriteLine(new string(' ', 10) + new string('-', 94));
            Console.Write("{0,10}|", "Item #");
            foreach (var category in tasks.Keys)
            {
                Console.Write("{0,30}|", category);
            }
            Console.WriteLine();
            Console.WriteLine(new string(' ', 10) + new string('-', 94));
            for(int i = 0; i < max; i++)
```

```
54              {
55                  Console.Write("{0,10}|", i+1);
56                  foreach(var list in tasks.Values)
57                  {
58                      if (list.Count > i)
59                          Console.Write("{0,30}|", list[i]);
60                      else
61                          Console.Write("{0,30}|", "N/A");
62                  }
63                  Console.WriteLine();
64              }
65              Console.ResetColor();
66          }
67
68          static void Main(string[] args)
69          {
70              var tasks = new Dictionary<string, List<string>>();
71              tasks["Personal"] = new List<string>();
72              tasks["Work"] = new List<string>();
73              tasks["Family"] = new List<string>();
74
75              string category;
76              string task;
77
78              while (true)
79              {
80                  Console.Clear();
81                  PrintTasks(tasks);
82
83                  category = ReadString("\nWhich category do you want to place " +
84                      "a new task? Type \'Personal\', \'Work\', \'Family\', or
                        ↪  \'Quit\'").ToLower();
85                  if (category.ToLower() == "quit")
86                      break;
87
88                  task = ReadString("Describe your task below (max. 30 symbols).");
89                  if (task.Length > 30)
90                  {
91                      task = task.Substring(0, 30);
92                  }
93
94                  try
95                  {
96                      tasks[category].Add(task);
97                  }
98                  catch (ArgumentException)
99                  {
100                     continue;  // if category is not present, add no task
101                 }
102             }
103         }
104     }
105 }
```

```cs
using System;
using System.Linq;
using System.Collections.Generic;

namespace DuplicateCode
{
    /// <summary>
    /// List of available options for actions in the program
    /// </summary>
    public enum MenuOption
    {
        AddCategory,
        DeleteCategory,
        AddTask,
        DeleteTask,
        CompleteTask,
        ChangeImportance,
        SetDueDate,
        ChangePosition,
        MoveTask,
        Quit
    }

    /// <summary>
    /// Level of task mportance. All overdue tasks print white on red
    /// background, regardless of the importance level.
    /// </summary>
    public enum TaskImportance
    {
        Low,
        Medium,
        High,
        Complete
    }

    /// <summary>
    /// Static class to provide methods for console input
    /// </summary>
    public static class ConsoleInput
    {
        /// <summary>
        /// Outputs a prompt for input and returns the input string
        /// <summary>
        /// <returns>
        /// The input of the user as a string
        /// </summary>
        /// <param name="prompt">The string to prompt the user with</param>
        public static string ReadString(string prompt)
        {
            Console.WriteLine(prompt);
            Console.Write(">> ");
            return Console.ReadLine();
        }
```

```
54
55          /// <summary>
56          /// Outputs a prompt for input and returns the input as an integer
57          /// <summary>
58          /// <returns>
59          /// The input of the user as an integer
60          /// </summary>
61          /// <param name="prompt">The string to prompt the user with</param>
62          public static int ReadInteger(string prompt)
63          {
64              string input = ReadString(prompt);
65              int output;
66              while (!int.TryParse(input, out output))
67              {
68                  Console.WriteLine("Enter a whole number only");
69                  input = ReadString(prompt);
70              }
71              return Convert.ToInt32(input);
72          }
73
74          /// <summary>
75          /// Outputs a prompt for input and returns the input as an integer
76          /// within a range from min to max
77          /// <summary>
78          /// <returns>
79          /// The input of the user as an integer
80          /// </summary>
81          /// <param name="prompt">The string to prompt the user with</param>
82          /// <param name="min">The minimum number allowed</param>
83          /// <param name="max">The maximum number allowed</param>
84          public static int ReadInteger(string prompt, int min, int max)
85          {
86              int input = ReadInteger(prompt);
87              while(input < min || input > max)
88              {
89                  Console.WriteLine("Enter a number between {0} to {1}", min, max);
90                  input = ReadInteger(prompt);
91              }
92              return input;
93          }
94      }
95
96      /// <summary>
97      /// Model for an individual task
98      /// </summary>
99      public class TaskModel
100     {
101         // Public properties
102         public string Description { get; set;}
103         public TaskImportance Importance { get; set; }
104         public DateTime DueDate { get; set; }
105
106         /// <summary>
```

```
107         /// Creates a new task object
108         /// </summary>
109         /// <param name="description">Description for the task</param>
110         /// <param name="importance">Level of task importance</param>
111         public TaskModel(
112             string description,
113             DateTime dueDate,
114             TaskImportance importance = TaskImportance.Medium)
115         {
116             Description = description;
117             Importance = importance;
118             DueDate = dueDate;
119         }
120 
121         /// <summary>
122         /// Returns the console color for the corresponding task
123         /// importance
124         /// </summary>
125         /// <returns>
126         /// ConsoleColor relevant to the importance
127         /// </returns>
128         public ConsoleColor GetColor()
129         {
130             switch (Importance)
131             {
132                 case TaskImportance.Low:
133                     return ConsoleColor.Green;
134                 case TaskImportance.Medium:
135                     return ConsoleColor.Blue;
136                 case TaskImportance.High:
137                     return ConsoleColor.Red;
138                 case TaskImportance.Complete:
139                 default:
140                     return ConsoleColor.DarkGray;
141             }
142         }
143     }
144 
145     /// <summary>
146     /// Repository for a collection of tasks within a category
147     /// </summary>
148     public class TaskList
149     {
150         // Public properties
151         private List<TaskModel> _tasks;
152 
153         /// <summary>
154         /// Creates a new task list
155         /// </summary>
156         public TaskList()
157         {
158             _tasks = new List<TaskModel>();
159         }
```

```
160
161         /// <summary>
162         /// Adds a task to the list of tasks
163         /// </summary>
164         /// <param name="task">The TaskModel to add to the list</param>
165         public void AddTask(TaskModel task)
166         {
167             _tasks.Add(task);
168         }
169
170         /// <summary>
171         /// Removes a task from the list of tasks
172         /// </summary>
173         /// <param name="task">The TaskModel to remove from the list</param>
174         public void DeleteTask(TaskModel task)
175         {
176             _tasks.Remove(task);
177         }
178
179         /// <summary>
180         /// Changes the position of a task within the task list
181         /// </summary>
182         /// <param name="currentIndex">Current index of the task</param>
183         /// <param name="newIndex">New index for the task</param>
184         public void ChangePriority(int currentIndex, int newIndex)
185         {
186             TaskModel task = new TaskModel(
187                 _tasks[currentIndex].Description,
188                 _tasks[currentIndex].DueDate,
189                 _tasks[currentIndex].Importance
190             );
191             _tasks.Remove(_tasks[currentIndex]);
192             _tasks.Insert(newIndex, task);
193         }
194
195         /// <summary>
196         /// Returns the count of tasks in the tasklist
197         /// </summary>
198         /// <returns>Integer of the number of tasks</returns>
199         public int GetCount()
200         {
201             return _tasks.Count;
202         }
203
204         /// <summary>
205         /// Returns the task at a specific index in range
206         /// </summary>
207         /// <returns>
208         /// The TaskModel at a specific index
209         /// </returns>
210         /// <param name="index">The index position of the task to return</param>
211         public TaskModel TaskAt(int index)
212         {
```

```
213             try
214             {
215                 return _tasks[index];
216             }
217             catch (IndexOutOfRangeException)
218             {
219                 throw;
220             }
221         }
222     }
223 
224     public class TaskListRepository
225     {
226         // Instance variables
227         private Dictionary<string, TaskList> _repo;
228 
229         /// <summary>
230         /// Creates a new TaskListRepository
231         /// </summary>
232         public TaskListRepository()
233         {
234             _repo = new Dictionary<string, TaskList>();
235         }
236 
237         /// <summary>
238         /// Adds a new category of tasks to the controller
239         /// </summary>
240         /// <param name="category"></param>
241         public void AddCategory(string category)
242         {
243             TaskList tasks = new TaskList();
244             _repo.Add(category, tasks);
245         }
246 
247         /// <summary>
248         /// Removes a category and associated tasks
249         /// </summary>
250         /// <param name="category"></param>
251         public void DeleteCategory(string category)
252         {
253             _repo.Remove(category);
254         }
255 
256         /// <summary>
257         /// Moves a task from one category to another
258         /// </summary>
259         /// <param name="task">The task to move</param>
260         /// <param name="current">Name of the current category</param>
261         /// <param name="updated">Name of the new category</param>
262         public void MoveTask(TaskModel task, string current, string updated)
263         {
264             _repo[current].DeleteTask(task);
265             _repo[updated].AddTask(task);
```

```
266         }
267
268         /// <summary>
269         /// Returns the length of the longest list in the dictionary
270         /// </summary>
271         /// <returns>
272         /// Length of the longest list as an integer
273         /// </returns>
274         public int MaximumLength()
275         {
276             return _repo.Values.Max(list => list.GetCount());
277         }
278
279         /// <summary>
280         /// Returns the task list for the given category
281         /// </summary>
282         /// <param name="category"></param>
283         /// <returns></returns>
284         public TaskList GetTaskList(string category)
285         {
286             return _repo[category];
287         }
288
289         /// <summary>
290         /// Returns whether a specific key exists in the repository
291         /// </summary>
292         /// <param name="key">The key to check for</param>
293         /// <returns>
294         /// Boolean whether the key exists or not
295         /// </returns>
296         public bool ContainsKey(string category)
297         {
298             return _repo.ContainsKey(category);
299         }
300
301         /// <summary>
302         /// Prints the current list of tasks by category, out to the
303         /// console
304         /// </summary>
305         public void Print()
306         {
307             Console.ForegroundColor = ConsoleColor.Blue;
308             Console.WriteLine(new string(' ', 12) + "CATEGORIES");
309             Console.WriteLine(new string(' ', 10)
310                 + new string('-', 51 * _repo.Count));
311             Console.Write("{0,10}|", "Item #");
312             foreach (var category in _repo.Keys) // Print the category names
313             {
314                 Console.Write("{0,-50}|", category);
315             }
316             Console.WriteLine();
317             Console.WriteLine(new string(' ', 10)
318                 + new string('-', 51 * _repo.Count));
```

```
            for(int i = 0; i < MaximumLength(); i++)
            {
                Console.Write("{0,10}|", i+1);
                foreach(var list in _repo.Values) // Print the list of tasks
                {
                    if (list.GetCount() > i)
                    {
                        Console.ForegroundColor = list.TaskAt(i).GetColor();
                        // Print white on red background if not complete and due
                        // today or overdue
                        if (list.TaskAt(i).DueDate <= DateTime.Today
                            && list.TaskAt(i).Importance != TaskImportance.Complete)
                        {
                            Console.BackgroundColor = ConsoleColor.Red;
                            Console.ForegroundColor = ConsoleColor.White;
                        }
                        Console.Write("{0,-30}{1,20}", list.TaskAt(i).Description,
                            list.TaskAt(i).DueDate.Date.ToString("d"));
                        Console.ResetColor();
                        Console.ForegroundColor = ConsoleColor.Blue;
                        Console.Write("|");
                    }
                    else
                        Console.Write("{0,-50}|", "N/A");
                }
                Console.WriteLine();
            }
            Console.ResetColor();
        }
    }

    /// <summary>
    /// Manages a collection of task lists
    /// </summary>
    public class TaskListController
    {
        // Instance variables
        private TaskListRepository _taskRepo;
        private MenuOption _action;

        /// <summary>
        /// Creates a new task list controller
        /// </summary>
        public TaskListController()
        {
            _taskRepo = new TaskListRepository();
        }

        /// <summary>
        /// Prints the menu options to the console
        /// </summary>
        static void PrintMenu()
        {
```

```
372             Console.WriteLine("\n" + new string('-', 30));
373             Console.WriteLine("| {0,-26} |", "MENU:");
374             Console.WriteLine(new string('-', 30));
375             Console.WriteLine("| {0,-26} |", " 1. Add Category");
376             Console.WriteLine("| {0,-26} |", " 2. Delete Category");
377             Console.WriteLine("| {0,-26} |", " 3. Add Task");
378             Console.WriteLine("| {0,-26} |", " 4. Delete Task");
379             Console.WriteLine("| {0,-26} |", " 5. Mark Task Complete");
380             Console.WriteLine("| {0,-26} |", " 6. Set Task Importance");
381             Console.WriteLine("| {0,-26} |", " 7. Set Task Due Date");
382             Console.WriteLine("| {0,-26} |", " 8. Change Task Position");
383             Console.WriteLine("| {0,-26} |", " 9. Move Task Category");
384             Console.WriteLine("| {0,-26} |", "10. Quit");
385             Console.WriteLine(new string('-', 30));
386         }
387 
388         /// <summary>
389         /// Selectas an action to perform from the menu
390         /// </summary>
391         /// <param name="repo">The TaskListController to tak action on</param>
392         /// <returns>MenuOption of the required option</returns>
393         static MenuOption ReadUserOption()
394         {
395             int action = ConsoleInput.ReadInteger("\nChoose a menu option:",
396                 1, Enum.GetNames(typeof(MenuOption)).Length);
397             return (MenuOption)action - 1;
398         }
399 
400         /// <summary>
401         /// Selects a valid category in the current repository of tasks
402         /// </summary>
403         /// <returns>The selected tasklist if the category exists</returns>
404         public TaskList SelectTaskList()
405         {
406             string category = ConsoleInput.ReadString("Enter name of category");
407             while(!_taskRepo.ContainsKey(category))
408             {
409                 Console.WriteLine("That is an invalid key");
410                 category = ConsoleInput.ReadString("Enter name of category");
411             }
412             return _taskRepo.GetTaskList(category);
413         }
414 
415         /// <summary>
416         /// Selects a task from a tasklist
417         /// </summary>
418         /// <param name="tasks"></param>
419         /// <returns></returns>
420         public TaskModel SelectTask(TaskList tasklist)
421         {
422             int selected = ConsoleInput.ReadInteger(
423                 "Task item #", 1, tasklist.GetCount());
424             return tasklist.TaskAt(selected - 1);
```

```
425         }
426
427         /// <summary>
428         /// Provides continuous looping of the controller, to maintain
429         /// and manage a collection of task lists while the user
430         /// continues to use the program.
431         /// </summary>
432         public void Run()
433         {
434             // Add initial default categories
435             _taskRepo.AddCategory("work");
436             _taskRepo.AddCategory("family");
437             _taskRepo.AddCategory("personal");
438
439             do
440             {
441                 Console.Clear();
442                 _taskRepo.Print();
443                 PrintMenu();
444
445                 _action = ReadUserOption();
446
447                 switch (_action)
448                 {
449                     case MenuOption.AddCategory:
450                         string category = ConsoleInput.ReadString(
451                             "Name of the category");
452                         if (!_taskRepo.ContainsKey(category))
453                             _taskRepo.AddCategory(category);
454                         break;
455
456                     case MenuOption.DeleteCategory:
457                         category = ConsoleInput.ReadString(
458                             "Name of the category");
459                         if (_taskRepo.ContainsKey(category))
460                             _taskRepo.DeleteCategory(category);
461                         break;
462
463                     case MenuOption.AddTask:
464                         TaskList tasklist = SelectTaskList();
465                         string description = ConsoleInput.ReadString(
466                             "Describe your task below (max. 30 symbols).");
467                         if (description.Length > 30)
468                         {
469                             description = description.Substring(0, 30);
470                         }
471                         DateTime dueDate = DateTime.Now;
472                         tasklist.AddTask(new TaskModel(description,
473                             dueDate.AddDays(1))); // Default due in 1 day
474                         break;
475
476                     case MenuOption.CompleteTask:
477                         tasklist = SelectTaskList();
```

```
478                         TaskModel task = SelectTask(tasklist);
479                         task.Importance = TaskImportance.Complete;
480                         break;
481
482                     case MenuOption.DeleteTask:
483                         tasklist = SelectTaskList();
484                         if (tasklist.GetCount() != 0)
485                         {
486                             task = SelectTask(tasklist);
487                             tasklist.DeleteTask(task);
488                         }
489                         break;
490
491                     case MenuOption.ChangeImportance:
492                         tasklist = SelectTaskList();
493                         task = SelectTask(tasklist);
494                         string importance = ConsoleInput.ReadString(
495                                 "Enter new importance");
496                         switch (importance.ToLower())
497                         {
498                             case "low":
499                                 task.Importance = TaskImportance.Low;
500                                 break;
501                             case "medium":
502                                 task.Importance = TaskImportance.Medium;
503                                 break;
504                             case "high":
505                                 task.Importance = TaskImportance.High;
506                                 break;
507                             case "complete":
508                                 task.Importance = TaskImportance.Complete;
509                                 break;
510                             default:
511                                 break;
512                         }
513                         break;
514
515                     case MenuOption.SetDueDate:
516                         tasklist = SelectTaskList();
517                         task = SelectTask(tasklist);
518                         string date = ConsoleInput.ReadString(
519                             "Enter due date \'YYYY-MM-DD\'");
520                         task.DueDate = DateTime.Parse(date);
521                         break;
522
523                     case MenuOption.ChangePosition:
524                         tasklist = SelectTaskList();
525                         int c = ConsoleInput.ReadInteger(
526                             "Enter task current item #");
527                         int n = ConsoleInput.ReadInteger(
528                             "Enter the new item #");
529                         tasklist.ChangePriority(c - 1, n - 1);
530                         break;
```

```
531
532                     case MenuOption.MoveTask:
533                         string current = ConsoleInput.ReadString(
534                             "Name of the current category");
535                         tasklist = _taskRepo.GetTaskList(current);
536                         task = SelectTask(tasklist);
537                         string updated = ConsoleInput.ReadString(
538                             "Name of the new category");
539                         _taskRepo.MoveTask(task, current, updated);
540                         break;
541
542                     case MenuOption.Quit:
543                     default:
544                         break;
545                 }
546             } while (_action != MenuOption.Quit);
547         }
548     }
549
550     /// <summary>
551     /// Final code for Task 4.2P
552     /// </summary>
553     class FinalCode
554     {
555         static void Main(string[] args)
556         {
557             var controller = new TaskListController();
558             controller.Run();
559         }
560     }
561 }
```