A Comprehensive Review of Eat The Frog: A Java Application

Kuljit Kaur Takhar

Harvard Extension School, Harvard University

CSCI E-10B: Introduction to Computer Science using Java II

Dr. Henry Leitner

David Habermehl

December 12, 2023

The Eat the Frog is a comprehensive task management tool built in Java that combines data structures, graphical user interface, components, methods, file management, and class design to provide users with an effective task management solution. Inspired by the time management strategy popularized by a quote attributed to Mark Twain, "Eat a live frog first thing in the morning, and nothing worse will happen to you the rest of the day," the application is designed to assist users in organizing tasks by priority.

The basic premise of the program is to enable users to add up to five tasks to a list sorted by priority level in non-decreasing order. The list should be easy modifiable. The input panel includes fields for task title, description, and a priority level dropdown-list. The list persists in the display window as well as in a text file that the application can read and write. A task can be edited, deleted, and marked complete. In the case of being marked complete, the task will be added to another text file for future reference.

The architecture of the program involves two classes and three data structures. The primary class involved in this application is EatTheFrog. This class serves as the core of the application, managing the graphical user interface, user interactions, and task-related operations. Users can add, edit, delete, and complete tasks via this class.

The class includes various methods. The addTask() method handles the addition of new tasks. It retrieves task details from user input fields, checks for duplicate priorities, and then creates a new Task object. This method ensures

that tasks are stored in the appropriate data structures and that the GUI is updated accordingly.

The editTask() method allows users to modify existing tasks. It manages both edit mode and saving changes to a task. Users can enter edit mode to update task details, and once changes are confirmed, the method ensures that the task is updated in the data structures, the GUI, and saved to the file.

The deleteTask() method enables users to remove selected tasks. It prompts users for confirmation and proceeds with deletion only when the user confirms. Deleted tasks are removed from both the data structures and the GUI, maintaining data integrity.

The completed() method allows users to mark tasks as completed. It displays a confirmation dialog to verify user intent. When confirmed, the method updates the task's completion status and takes appropriate action, such as moving the completed task to a separate list or file.

Another aspect of the EatTheFrog class is file management. The file management functionalities within the application that read and write files ensure the persistence and accessibility of task data. Leveraging file streams, particularly BufferedReader and BufferedWriter, this aspect of the application handles reading from and writing to a file named eatthefrog.txt using the method loadTasksFromFile(). During the application's initialization, the program checks for the existence of this file. If found, it employs the BufferedReader to systematically retrieve task details from each line in the file. These details,

including title, description, priority, and completion status, are then used to reconstruct the corresponding Task objects. Simultaneously, the tasks are added to both the application's in-memory data structures, namely the taskMap and taskSet, and the user interface's taskListModel for immediate display. This process of data loading ensures that users can easily resume managing their tasks from where they left off. If the file does not exist, the application will create the eatthefrog.txt file when a user adds the first task, ensuring data persistence from that point onwards. This approach to file management bolsters data integrity, providing a dimensional user experience and making Eat the Frog a reliable tool for efficient task organization.

The other class implemented in the program is the Task class. The Task class represents the individual tasks within the application. This class encapsulates task attributes like title, description, priority, and completion status. Task data is stored and sorted efficiently using the Map and TreeSet data structures. Additionally, the program implements the Comparable interface to enable sorting based on task priority. The TaskComparator class is not explicitly defined in the code but is used by the TreeSet data structure to ensure tasks are sorted automatically.

Eat the Frog utilizes three main data structures: Map, TreeSet, and ArrayList. The application's core data structure, a Map named taskMap, plays a pivotal role in efficiently managing tasks. This Map stores tasks as values, associating each task with a unique integer key based on its hash code. Tasks

can be added, edited, and deleted by manipulating this Map, ensuring easy access, modifications, and updates. The use of the map enhances the application's performance and allows users to interact with their tasks effortlessly.

Additionally, Eat the Frog heavily relies on the TreeSet data structure named taskSet to prioritize tasks efficiently. A TreeSet is chosen for its automatic sorting capabilities. In this context, the Task class has been implemented to be comparable, enabling tasks to be sorted based on their user-programmed priority level. This choice of data structure ensures that tasks are always presented to the user in the correct order of priority, a crucial aspect of this specific task management philosophy. A TreeSet is essentially an implementation of a self-balancing binary search tree. When new tasks are added or existing ones are edited to change their priorities, the TreeSet automatically reorganizes the tasks, eliminating the need for manual sorting operations. The TreeSet then copies the data into an ArrayList named taskList.

The application also utilizes the ArrayList data structure. In the sortTaskListModel() method, which is called to sort the list of tasks based on priority, the tasks are copied to an ArrayList and then re-populated in the list model. By using a combination of Map, TreeSet, and ArrayList, the application optimizes the user experience by consistently displaying tasks according to their importance, helping users focus on high-priority tasks first.

The capabilities of Eat the Frog extend beyond task management and prioritization. File management functionality has been embedded within the EatTheFrog class. It enhances the application's usability by offering the capability to load and save tasks effectively. To achieve this, BufferedReader and BufferedWriter are employed as file streams to read tasks from and write tasks to a file named eatthefrog.txt. During the initialization of the application, the program checks for the existence of this file. If the file is found, the BufferedReader is utilized to systematically retrieve task details from each line in the file. These details encompass essential task attributes such as title, description, priority, and completion status. These attributes are used to reconstruct the corresponding Task objects, which are subsequently added to both the in-memory data structures and the user interface's taskListModel. This process ensures that users can quickly resume managing their tasks, with all the pertinent data readily available. Furthermore, the application is intelligent enough to create the eatthefrog.txt file when a user adds their first task. This dynamic file management approach ensures data persistence and reliability, even for new users. Thus, users can trust that their tasks are securely stored and can be accessed whenever needed.

An additional functionality utilizing file management is creating and maintaining a text file called frogseaten.txt wherein the tasks that have been completed are then added to the file using saveTasksToFile() in order to maintain a record of completed tasks. Studies also show that keeping a record of

completed tasks can serve as a motivational factor as well as increase overall life satisfaction. When the task is removed from the GUI after selecting completed, it will persist in an easily readable format.

The graphical user interface of Eat the Frog utilizes a range of Swing components that provide an intuitive and visually harmonious user experience. The main window of the application is encapsulated within a JFrame. This top-level container not only provides a platform for other GUI components but also manages the overall layout and appearance of the application window. It ensures that users can easily access and navigate through the various features and functionalities offered by Eat the Frog. The heart of the task management GUI of the application lies within a JList. This component serves as a dynamic list that displays all the tasks currently managed by the application. Users can view their tasks at a glance, and the list automatically updates to reflect any changes, such as the addition, editing, deletion, or completion of tasks. The use of a JList ensures that task management remains organized and accessible.

The JTextField and JTextArea components are instrumental in enabling users to input and edit task details. JTextField is used to input task titles, while JTextArea creates a field for task descriptions. The application employs JButton to create a responsive user interface. Buttons like "Add," “Edit," and “Delete," are integral to the applications use and functionality by performing the corresponding methods. The "Completed" checkbox allows users the catharsis of checking a box and removing an item from your list while copying that task

into a text file. Users can easily trigger these actions for an intuitive UI experience.

The JComboBox component plays a crucial role in the task prioritization function of this program. Users can select the priority level of each task from a drop-down menu provided by the JComboBox. The available priority levels range from 1 (highest) to 5 (lowest). This feature enables users to quickly categorize and order their tasks by importance. The JCheckBox allows users to mark tasks as completed with a simple click. This checkbox functionality streamlines the task tracking process, providing a clear indication of a task's status. The Swing components are able to perform these actions through event-driven programming.

Eat the Frog adopts event-driven programming, meaning that it responds to user actions by triggering specific event handlers. For instance, when a user clicks the "Add" button after entering a task, the application responds to this event by invoking the corresponding event handler, initiating the task creation process. This event-driven approach ensures that the application remains responsive and interactive, catering to the dynamic nature of task management.

The layout of the GUI is structured using a GridLayout for the input panel, ensuring a clear organized arrangement of input fields for task details. This grid-based layout maintains consistency and modular aesthetics, making it easier for users to input information. Additionally, the use of colors, fonts, and alignment enhances the visual appeal of the interface and the user experience.

The program also improves user experience through its handling of exceptions by implementing try-catch blocks in various methods to ensure that user input is valid and that file operations are handled gracefully. For example, when adding a new task using the addTask() method, it catches a NumberFormatException if there is an issue parsing the priority from the combo box. In such cases, it displays a dialog to inform the user about the invalid input. Similarly, the editTask() method and the deleteTask() method include checks for the selected task's index to ensure it's a valid selection. If the index is out of bounds, a dialog notifies the user to select a task before performing the action. When loading tasks from a file using the loadTasksFromFile() method and saving tasks to a file with the saveTasksToFile() method, the program handles potential IOExceptions by printing the stack trace but does not interrupt the program's execution. Through the use of exception handling, users receive clear feedback and instructions when they attempt actions that are not permitted. The try-with-resources statement has the added benefit of automatically closing the BufferedReader when you exit the try block, eliminating data leakage.

A use case of the application is solving the task management problem of where-to-begin. A user overwhelmed with tasks opens the program and enters the task details of a particular objective into the task and description field. The user must then select a priority number. The user then adds subsequent tasks and assigns priority levels to those tasks as well. Priority levels cannot be duplicated. When a user tries to duplicate a priority level, they will receive an

error message explaining the action is not permitted. The user edits the priority levels as they think of other tasks and reconsider priorities. The program reorders the list accordingly. Once the user gets to five tasks, the list is full. The user can delete tasks and add new tasks but the list will never exceed five, giving the user a manageable amount of tasks to complete. The user can now begin completing their tasks starting with the top of the list, the item with the highest priority. As the user completes tasks, the tasks are removed from the window display and written into a separate file. The user may reference the file in the future for motivation and fulfillment.

In conclusion, the Eat The Frog Java application demonstrates effective utilization of the data structures Map for task management and TreeSet and ArrayList for sorting, a user-friendly Swing interface, and well-defined methods across multiple classes. The application's file management capabilities ensure that task data is persisted between sessions. The class design, comprising the EatTheFrog and Task classes, supports modularity and encapsulation, enhancing code readability and maintainability. The application optimizes task prioritization making it a valuable tool for efficient task management.