Relatório Detalhado do Funcionamento do Código

Bruno Watanabe, Eduardo Civitate, Gabriel Palhares

Sumário

1 Visão Geral do Projeto **2**

2 Principais Classes e Pacotes **2**

2.1 Pacote `org.example` 2

2.2 Pacote `org.example.interfaceGrafica` 3

2.3 Pacote `org.example.memento` 4

2.4 Pacote `org.example.task` 5

2.5 Pacote `org.example.task.manager` 6

2.6 Pacote `org.example.thread` 7

3 Testes e Qualidade do Código **7**

4 Fluxo de Execução e Diagramas de Sequência **8**

4.1 Adicionar Tarefa 8

4.2 Remover Tarefa 9

5 Padrões de Projeto Utilizados **10**

6 Conclusão **10**

1 Visão Geral do Projeto

Este projeto implementa um gerenciador de lista de tarefas (TO-DO list) em Java, com as seguintes funcionalidades:

- Criação, remoção, edição e marcação de tarefas como concluídas ou não.
- Persistência de dados em arquivos, permitindo manter as tarefas entre execuções.
- Suporte a múltiplos usuários, cada um com seu próprio arquivo de tarefas.
- Utilização de padrões de projeto GoF: **Memento** (para Undo), **Factory Method** (para criação de tarefas) e **Observer** (para atualizar a interface gráfica quando as tarefas mudam).
- Interface gráfica desenvolvida com Swing.
- Salvamento automático periódico (AutoSave) a cada 15 segundos.
- Funcionalidade de "Desfazer"(Undo) através do padrão Memento.

2 Principais Classes e Pacotes

2.1 Pacote `org.example`

Main: Contém o método `main`, ponto de entrada do programa. Ele pergunta o nome do usuário, carrega as tarefas correspondentes (por exemplo, `joao_tasks.db`), cria a interface gráfica (`MainFrame`) e inicia a `AutoSaveThread`.

Exemplo de código (simplificado):

```
1  public class Main {
2      public static void main(String[] args) {
3          String username = JOptionPane.showInputDialog("Qual    o seu nome?"
               );
4          if (username == null || username.trim().isEmpty()) {
5              username = "usuario_padrao";
6          }
7
8          TaskManager taskManager = new TaskManager();
9          TaskManagerHistory history = new TaskManagerHistory();
10         TaskFactory factory = new SimpleTaskFactory();
11         File dbFile = new File(username + "_tasks.db");
12
13         // Carrega tarefas do arquivo do usu rio
14         if (dbFile.exists()) {
15             try {
```

```
16                  List<Task> loaded = TaskFileHandler.loadTasks(dbFile);
17                  for (Task t : loaded) {
18                      taskManager.addTask(t);
19                  }
20              } catch (IOException e) {
21                  System.err.println("N o foi poss vel carregar as tarefas
                       do usu rio " + username + ": " + e.getMessage());
22              }
23          }
24
25          SwingUtilities.invokeLater(() -> {
26              MainFrame frame = new MainFrame(taskManager, history, factory,
                    username);
27              taskManager.refreshObservers();
28
29              AutoSaveThread autoSaveThread = new AutoSaveThread(taskManager,
                     dbFile);
30              autoSaveThread.start();
31
32              frame.addWindowListener(new WindowAdapter() {
33                  @Override
34                  public void windowClosing(WindowEvent e) {
35                      try {
36                          TaskFileHandler.saveTasks(taskManager.getTasks(),
                               dbFile);
37                      } catch (IOException ex) {
38                          System.err.println("Erro ao salvar as tarefas de "
                               + username + ": " + ex.getMessage());
39                      }
40                      autoSaveThread.stopAutoSave();
41                  }
42              });
43
44              frame.setLocationRelativeTo(null);
45              frame.setVisible(true);
46          });
47      }
48  }
```

2.2 Pacote `org.example.interfaceGrafica`

MainFrame: Estende `JFrame` e implementa `TaskManagerObserver`. Esta é a janela principal da aplicação.

Principais funcionalidades:

- Ao adicionar, remover, editar ou marcar tarefas, o estado anterior é salvo no histórico

(`TaskManagerHistory`) para permitir Undo.

- Ao detectar mudanças no `TaskManager`, a interface é atualizada (padrão Observer).

Exemplo de métodos:

```
1  private void addTask() {
2      String title = JOptionPane.showInputDialog(this, "T tulo da tarefa:");
3      if (title != null && !title.isEmpty()) {
4          String desc = JOptionPane.showInputDialog(this, "Descri  o da
               tarefa:");
5          history.saveState(taskManager.createMemento()); // salva estado p/
               Undo
6          Task t = factory.createTask(title, desc == null ? "" : desc);
7          taskManager.addTask(t);
8      }
9  }
10
11 private void undo() {
12     Memento m = history.undo();
13     if (m != null) {
14         taskManager.restoreMemento(m); // restaura estado anterior
15     } else {
16         JOptionPane.showMessageDialog(this, "N o h  mais opera  es para
               desfazer.");
17     }
18 }
```

Notificação de mudanças:

```
1  @Override
2  public void onTasksChanged() {
3      listModel.clear();
4      for (Task t : taskManager.getTasks()) {
5          listModel.addElement((t.isDone() ? "[X] " : "[ ] ")
6            + t.getTitle() + " - " + t.getDescription());
7      }
8  }
```

2.3 Pacote `org.example.memento`

Memento: Armazena o estado (lista de `Task`) em um momento específico, permitindo restaurá-lo para Undo.

```
1  public class Memento {
2      private final List<Task> state;
3      public Memento(List<Task> state) { this.state = state; }
```

```
    public List<Task> getState() { return state; }
}
```

TaskManagerHistory: Gerencia o histórico de estados para implementar Undo.

```
public class TaskManagerHistory {
    private Stack<Memento> undoStack = new Stack<>();
    private Stack<Memento> redoStack = new Stack<>();

    public void saveState(Memento m) {
        undoStack.push(m);
        redoStack.clear();
    }

    public Memento undo() {
        if (!undoStack.isEmpty()) {
            Memento m = undoStack.pop();
            redoStack.push(m);
            return m;
        }
        return null;
    }

    public Memento redo() {
        if (!redoStack.isEmpty()) {
            Memento m = redoStack.pop();
            undoStack.push(m);
            return m;
        }
        return null;
    }
}
```

2.4 Pacote `org.example.task`

Task (Interface): Define métodos para acessar e modificar título, descrição e estado (concluída ou não).

```
public interface Task {
    String getTitle();
    void setTitle(String title);
    String getDescription();
    void setDescription(String description);
    boolean isDone();
    void setDone(boolean done);
}
```

SimpleTask: Implementação simples de `Task`.

TaskFactory: Classe abstrata para criar tarefas (Factory Method).

TaskFileHandler: Responsável por salvar e carregar tarefas do arquivo. Inclui tratamento de exceções (`IOException`) ao ler ou escrever o arquivo, informando o usuário sobre erros através de mensagens (ex: `System.err.println` ou `JOptionPane.showMessageDialo`

```
1  public static void saveTasks(List<Task> tasks, File file) throws
       IOException {
2      try (BufferedWriter bw = new BufferedWriter(new FileWriter(file))) {
3          for (Task t : tasks) {
4              bw.write(t.getTitle() + ";" + t.getDescription() + ";" + t.
                   isDone());
5              bw.newLine();
6          }
7      }
8  }
```

2.5 Pacote `org.example.task.manager`

TaskManager: Gerencia a lista de tarefas, notifica observadores quando há mudanças e cria/restaura Mementos.

```
1  public class TaskManager {
2      private List<Task> tasks = new ArrayList<>();
3      private List<TaskManagerObserver> observers = new ArrayList<>();
4
5      public void addTask(Task t) {
6          tasks.add(t);
7          notifyObservers();
8      }
9
10     public Memento createMemento() {
11         return new Memento(new ArrayList<>(tasks));
12     }
13
14     public void restoreMemento(Memento m) {
15         this.tasks = new ArrayList<>(m.getState());
16         notifyObservers();
17     }
18
19     private void notifyObservers() {
20         for (TaskManagerObserver obs : observers) {
21             obs.onTasksChanged();
22         }
23     }
24 }
```

TaskManagerObserver: Interface para observadores que implementam `onTasksChanged()`.

2.6 Pacote `org.example.thread`

AutoSaveThread: Salva as tarefas a cada 15 segundos.

```
1  public class AutoSaveThread extends Thread {
2      private final TaskManager taskManager;
3      private final File file;
4      private volatile boolean running = true;
5
6      public void run() {
7          while(running) {
8              try {
9                  Thread.sleep(15000);
10                 TaskFileHandler.saveTasks(taskManager.getTasks(), file);
11             } catch (InterruptedException e) {
12                 running = false;
13             } catch (IOException e) {
14                 System.err.println("Erro ao realizar auto-save: " + e.
                       getMessage());
15             }
16         }
17     }
18
19     public void stopAutoSave() {
20         running = false;
21         this.interrupt();
22     }
23 }
```

—

3 Testes e Qualidade do Código

Foram criados testes unitários para o `TaskManager` utilizando `JUnit`. Isso garante que as operações de adicionar, remover e atualizar tarefas funcionem corretamente.

```
1 @Test
2 public void testAddTask() {
3     Task t = new SimpleTask("Teste", "Desc");
4     tm.addTask(t);
5     assertEquals(1, tm.getTasks().size());
6     assertEquals("Teste", tm.getTasks().get(0).getTitle());
7 }
```

—

4 Fluxo de Execução e Diagramas de Sequência

A seguir, apresentamos dois diagramas de sequência que ilustram o fluxo de execução ao adicionar e ao remover uma tarefa, detalhando a comunicação entre `MainFrame`, `JOptionPane`, `TaskManager`, `Memento`, `TaskManagerHistory`, `TaskFactory` e `Task`.

4.1 Adicionar Tarefa

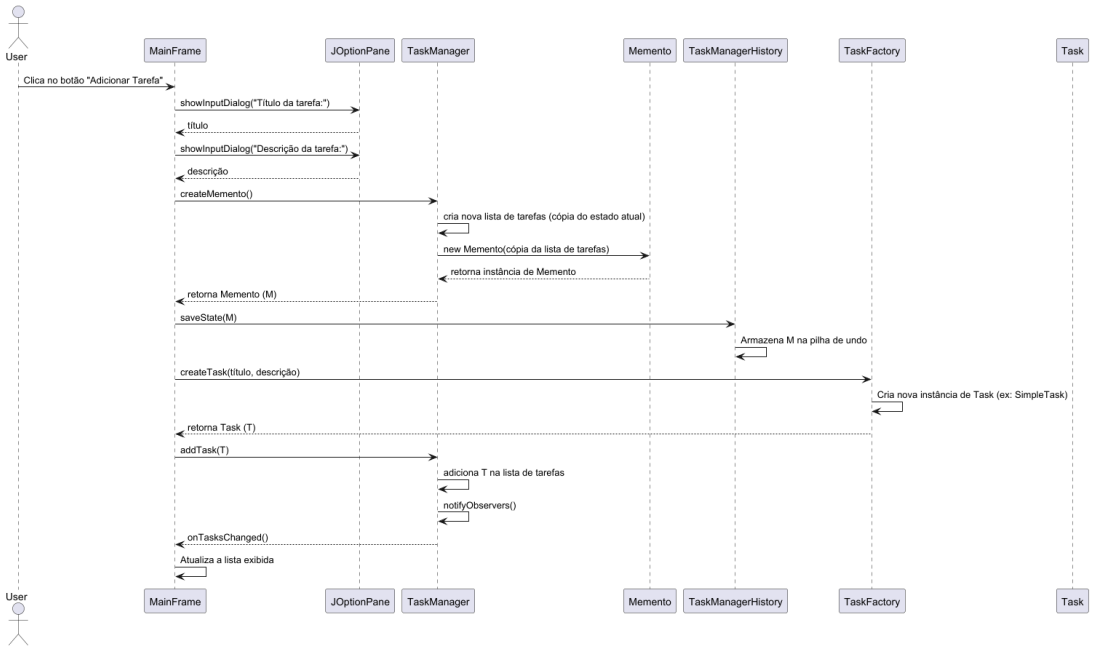

Figura 1: Diagrama de sequência ao adicionar uma tarefa.

Passo a Passo:

1. O usuário clica no botão "Adicionar Tarefa"na interface (`MainFrame`).
2. O `MainFrame` pede ao usuário o título e a descrição da tarefa via `JOptionPane`.
3. Antes de criar a nova tarefa, o `MainFrame` solicita ao `TaskManager` um memento do estado atual, chamando `createMemento()`. O `TaskManager` cria uma cópia da lista atual de tarefas e instancia um `Memento`.
4. O `MainFrame` chama `saveState(M)` no `TaskManagerHistory`, armazenando o memento.
5. O `MainFrame` solicita à `TaskFactory` a criação da nova tarefa.
6. A tarefa criada é adicionada ao `TaskManager` via `addTask(T)`.

7. O `TaskManager`, após modificar a lista de tarefas, notifica seus observadores (padrão Observer). O `MainFrame`, como observador, é avisado via `onTasksChanged()` e atualiza a interface.

4.2 Remover Tarefa

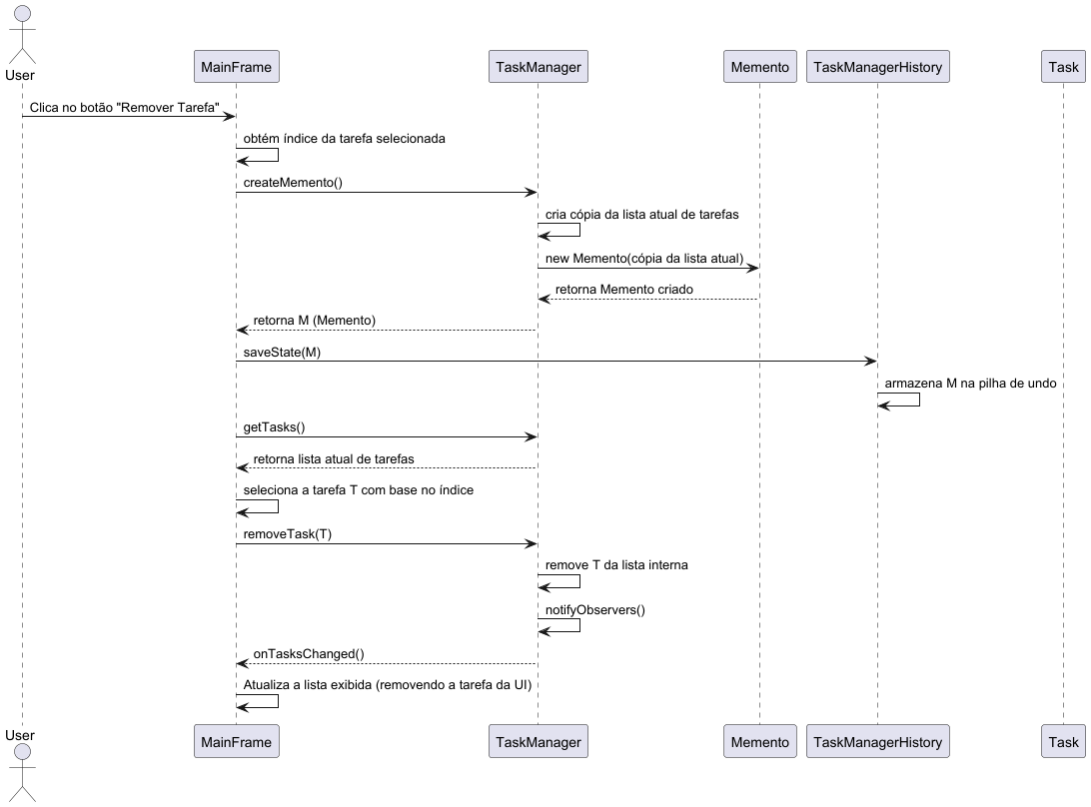

Figura 2: Diagrama de sequência ao remover uma tarefa.

Passo a Passo:

1. O usuário clica no botão "Remover Tarefa"no `MainFrame`.
2. O `MainFrame` obtém o índice da tarefa selecionada.
3. Antes de remover, o `MainFrame` chama `createMemento()` no `TaskManager` para obter um memento do estado atual, garantindo possibilidade de Undo.
4. O memento é salvo no `TaskManagerHistory` via `saveState(M)`.
5. O `MainFrame` obtém a lista de tarefas e seleciona a tarefa a remover.
6. A tarefa é removida via `removeTask(T)` no `TaskManager`.
7. O `TaskManager` notifica os observadores. O `MainFrame`, ao receber `onTasksChanged()`, atualiza a interface removendo a tarefa da visualização.

Esses diagramas mostram claramente a interação entre as classes e como os padrões Memento (para Undo), Factory Method (para criação de tarefas) e Observer (para atualização da UI) se integram no fluxo de execução.

—

5 Padrões de Projeto Utilizados

- **Memento (Undo)**: O `TaskManager` cria e restaura estados (Memento), permitindo desfazer operações.
- **Factory Method (Criação de Tarefas)**: A criação de novas tarefas é delegada a uma fábrica abstrata, facilitando a extensão do sistema.
- **Observer (Atualização da UI)**: O `MainFrame` é um observador do `TaskManager`. Quando as tarefas mudam, o `MainFrame` é notificado e atualiza a interface.

—

6 Conclusão

Este projeto demonstra o uso prático de padrões de projeto (Memento, Factory Method, Observer), persistência de dados, interface gráfica e threads, resultando em um sistema de lista de tarefas robusto, extensível e amigável ao usuário. Além disso, o tratamento de exceções garante que o usuário seja notificado adequadamente em caso de erros de IO, evitando falhas silenciosas. Os testes unitários com JUnit asseguram a confiabilidade das operações principais do `TaskManager`, dando maior segurança na evolução e manutenção do código.

Os diagramas de sequência apresentados detalham o fluxo de execução no momento de adicionar e remover tarefas, evidenciando a interação entre as classes e a utilização dos padrões de projeto adotados.