



PATRICK OLIVEIRA DE PAULA RA: 11002616
LUISA SALLES DE OLIVEIRA RA: 11201720136
ERIK COPEL ROTHMAN RA: 11018516

Gerenciador de Atividades Baseado na Técnica Pomodoro

Relatório apresentado à Universidade Federal do ABC como parte dos requisitos para aprovação na disciplina Programação Orientada a Objetos do Curso de Bacharelado em Ciência da Computação.

Professor: Prof. Vera Nagamuta

Santo André - SP

2019

Sumário

1 Introdução	3
2 Descrição geral	4
3 Descrição dos usuários	7
4 Tutorial de Uso	7
5 Implementação	12
6 Conclusões	20
7 Referências Bibliográficas	22

1 Introdução

A técnica pomodoro é uma técnica de produtividade e gerenciamento de tempo desenvolvida por Francesco Cirillo na década de 80. Nesta técnica, são especificados intervalos de 25 minutos para trabalho focado, seguidos de intervalos de descansos de 5 minutos, com um período maior de descanso após um número determinado de ciclos (ou pomodoros) – usualmente 15 minutos de descanso após 4 pomodoros. A duração de um pomodoro é uma decisão arbitrária e o padrão de 25 minutos é apenas um consenso mantido desde sua concepção. Entretanto, estudos mostram que a capacidade de concentração de uma pessoa em uma tarefa decresce com o tempo (*Davies & Parasuraman, 1982*), e uma possível solução consiste em estabelecer intervalos de descanso ou distração da tarefa central, com resultados demonstrando que isso contribui para estender o intervalo em que uma pessoa consegue se manter concentrada em uma tarefa (*Ariga & Lleras, 2011*).

A técnica é popular entre estudantes, músicos e profissionais. Pode-se entendê-la também como uma formulação mais simples da técnica de “*timeboxing*” para gerenciamento de processos, em que se divide a execução de um projeto em janelas de tempo, cada uma com especificações do que deve ser feito, duração máxima e custo (em um contexto empresarial). Seja para planejamento pessoal ou para grandes projetos com um grupo de participantes, esse tipo de gerenciamento de tempo pode ser consideravelmente beneficiado pelo uso de aplicativos.

Muitos aplicativos para gerenciamento de tempo e inspirados na técnica pomodoro já foram produzidos, dos mais simples, como um simples Timer para pomodoros de 25 minutos, aos mais complexos, para gerenciamento de projetos com grandes times. Entretanto, nenhum aplicativo se adequa plenamente às necessidades individuais, de modo que podem faltar funcionalidades ou a complexidade demasiada afasta usuários.

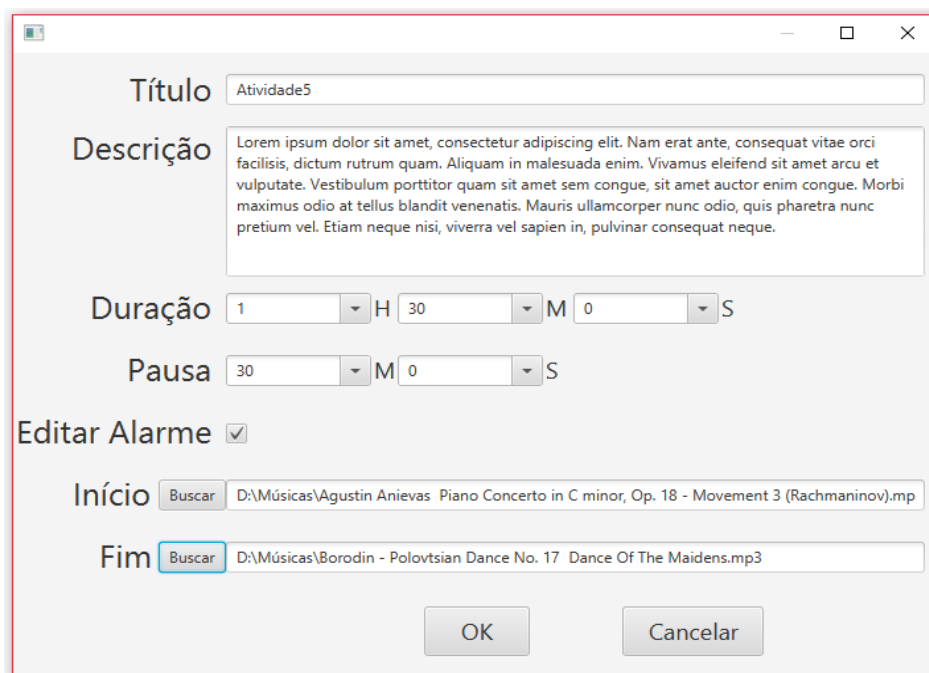
O objetivo do projeto é, assim, construir um sistema simples e intuitivo de gerenciamento de tempo inspirado na técnica pomodoro, possibilitando ao usuário criar listas de atividades com descrição e tempo de duração especificados, que serão executadas automaticamente em um timer, e organizando-as em diferentes perfis de uso que reúnem atividades similares. Dois tipos de atividades podem ser criadas: uma simples, com apenas descrição e tempo de duração, e uma lista de tarefas, em que o

usuário pode detalhar melhor uma atividade, dividindo-a em diferentes subtarefas, cada uma com uma expectativa de duração, que serão processadas automaticamente pelo aplicativo.

2 Descrição geral

A aplicação foi construída utilizando uma interface gráfica para viabilizar uma interação mais intuitiva e amigável com o usuário.

Até o momento, dois tipos de atividades foram implementadas, um tipo padrão, denominado simplesmente de “Atividade”, e uma ToDo List, ou lista de tarefas. Devido às diferenças de especificação das atividades, cada uma demanda uma interface gráfica de criação particular.



A janela de criação e edição de uma atividade simples apresenta os seguintes elementos:

- Título:** Campo de texto com o valor "Atividade5".
- Descrição:** Área de texto com conteúdo de preenchimento (Lorem ipsum).
- Duração:** Campos para horas (1), minutos (30) e segundos (0).
- Pausa:** Campos para minutos (30) e segundos (0).
- Editar Alarme:** Opção marcada com um checkbox.
- Início:** Campo de texto com o caminho de arquivo "D:\Músicas\Agustin Anievas Piano Concerto in C minor, Op. 18 - Movement 3 (Rachmaninov).mp" e um botão "Buscar".
- Fim:** Campo de texto com o caminho de arquivo "D:\Músicas\Borodin - Polovtsian Dance No. 17 Dance Of The Maidens.mp3" e um botão "Buscar".
- Botões:** "OK" e "Cancelar" na base da janela.

Figura 1 - Janela de criação e edição de uma atividade simples.

Uma atividade simples deve conter, no mínimo, um título, um tempo de duração e um tempo de pausa. A descrição e a customização dos alarmes são opcionais. Pode-se escolher dois alarmes, um de início, que será tocado quando uma atividade começa a ser executada, e um de término, que toca ao final da execução. No caso em que o usuário não escolhe customizá-los, um alarme padrão é escolhido.

Figura 2 Janela de criação e edição de uma ToDo List (lista de tarefas).

Uma lista de tarefas deve conter um título e pelo menos uma tarefa, com um número máximo de tarefas arbitrário. É necessário especificar o título e o número de tarefas antes que seja possível criá-las, apertando o botão “Adicionar Tarefas”. Até o momento não foram implementados métodos para editar a lista de tarefas, uma vez que foram criadas. Como na atividade simples, uma descrição e a customização dos alarmes é opcional.

As atividades criadas pelo usuário são apresentadas em uma janela principal, com duas listas e um painel central (Figura 3). A lista à esquerda contém os títulos das atividades do usuário, e ao selecionar qualquer um dos itens, suas informações de tempo e descrição são apresentadas no painel central. A lista à direita contém a seleção de atividades que o usuário escolhe para execução, clicando duas vezes em um nome da sua lista de atividades. É possível, clicando nos botões “Subir” e “Descer”, mudar a ordem de execução antes de iniciar o timer (ainda que seja possível alterar essa ordem de execução, com restrições, durante o funcionamento do contador), e também remover uma atividade da lista de preparação clicando duas vezes em uma atividade.

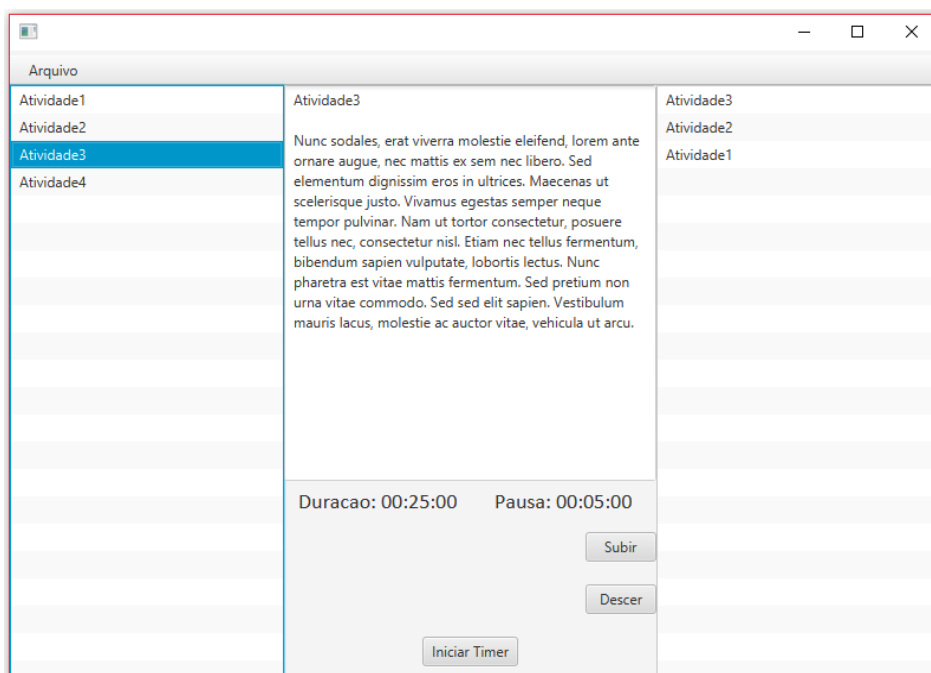


Figura 3 Janela principal de apresentação de atividades e seleção para execução.

Por fim, ao clicar em “Iniciar Timer”, a janela do contador é aberta, e seu funcionamento se dá de maneira automática. A execução segue um padrão de fila, executando o primeiro elemento escolhido (último da lista), até o último. Entretanto, o usuário pode pausar e resumir o contador, finalizar a atividade que está sendo executada, remover atividades da fila e mudar sua ordem de execução.

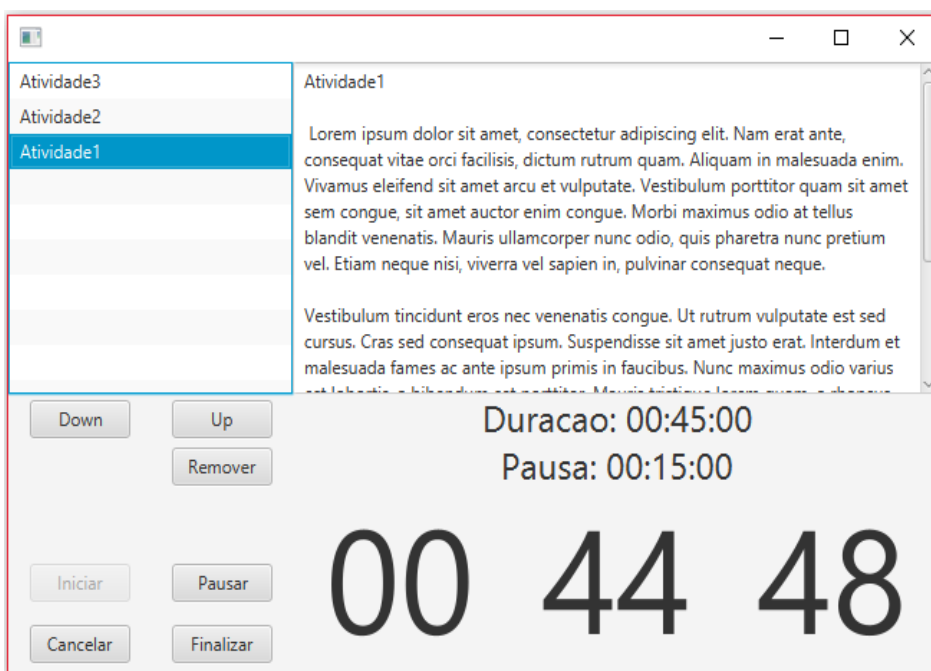


Figura 4 Janela de execução do timer.

3 Descrição dos usuários

Os usuários do sistema não são divididos em grupos com diferentes restrições de acesso. A opção de criação de perfis de usuário tem o propósito de possibilitar a criação de perfis de uso, isto é, a criação, para uma mesma pessoa, de diferentes perfis que reunirão atividades similares.

O leque de interesses que poderiam tirar proveito da aplicação é bastante amplo, e podemos citar alguns exemplos.

Uma estratégia de organização de estudos consiste em especificar para cada disciplina um número de horas de estudo que serão alocados ao longo da semana. Essas horas podem ser usadas para resolver um conjunto de obrigações, como a escrita de um artigo, resolução de exercícios ou a leitura de um livro. Pode-se criar um perfil de usuário contendo uma lista de pomodoros, um para cada uma dessas atividades, com suas durações e a descrição do que deve ser feito. Assim, ao longo da semana basta seguir a lista de atividades.

No estudo de música, costuma-se dividir o tempo em um período de estudo de teoria musical e outro de técnica para um instrumento musical. O estudo de técnica tem o objetivo de habituar os movimentos das mãos, a fim de alcançar maior velocidade, flexibilidade, clareza e acurácia, e existem exercícios específicos para cada capacidade. Uma sessão de estudos pode incluir um conjunto de exercícios com um tempo de duração pré-definido (geralmente não muito longo) e intervalos de descanso entre um exercício e outro (para evitar lesões). Para esse contexto, é bastante útil ter uma aplicação que automatize a organização dessa prática, e pode-se criar um perfil de usuário que reúna os pomodoros destes exercícios.

4 Tutorial de Uso

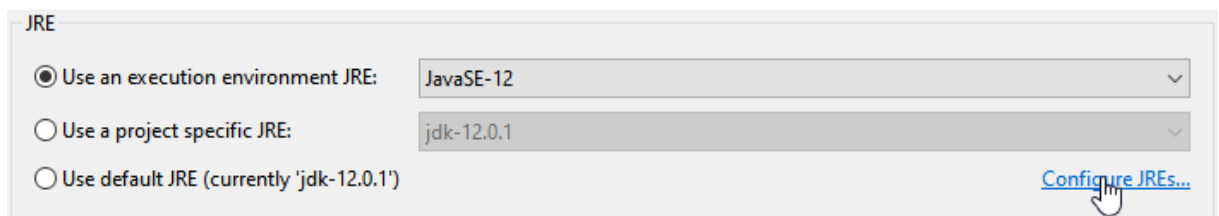
Para utilizar o arquivo executável (.jar) é necessário ter instalado no seu computador a versão 8 do Java, que pode ser encontrado no endereço https://www.java.com/pt_BR/download/.

Coloque o arquivo executável onde preferir, navegue pelo terminal até este diretório e o execute pelo comando “java -jar timer.jar”.

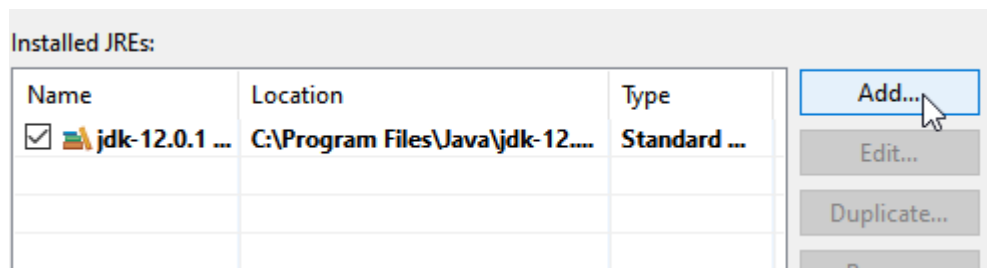
Para abrir o programa através do eclipse, é necessário fazer a instalação do Java Development Kit 12 e da biblioteca JavaFX. O tutorial de uso oficial dos

desenvolvedores do JavaFX pode ser encontrado no endereço <https://openjfx.io/openjfx-docs/>. De qualquer forma, providenciamos um tutorial de setup do projeto no Eclipse.

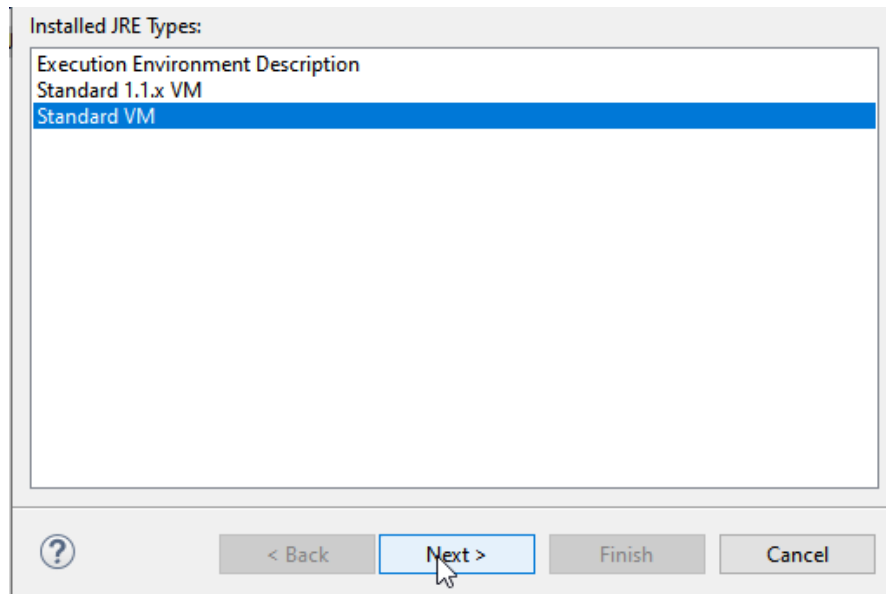
1. Acesse o link <https://jdk.java.net/12/> e baixe o pacote do Java Development Kit para o seu sistema operacional, e descompacte-o onde preferir. O arquivo contém uma pasta, então basta selecionar a opção de “descompactar aqui”.
2. Acesse o link <https://gluonhq.com/products/javafx/> e baixe o pacote “JavaFX ... SDK” versão 12 (em Latest Release) para o seu sistema operacional. Descompacte-o no seu diretório de preferência.



3. No eclipse, crie um novo projeto, e no espaço “JRE”, clique em “Configure JREs...”



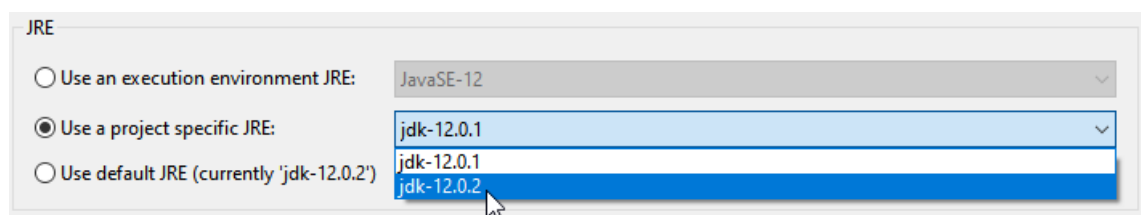
4. Em seguida, ao abrir uma nova janela, clique em “Add...”.



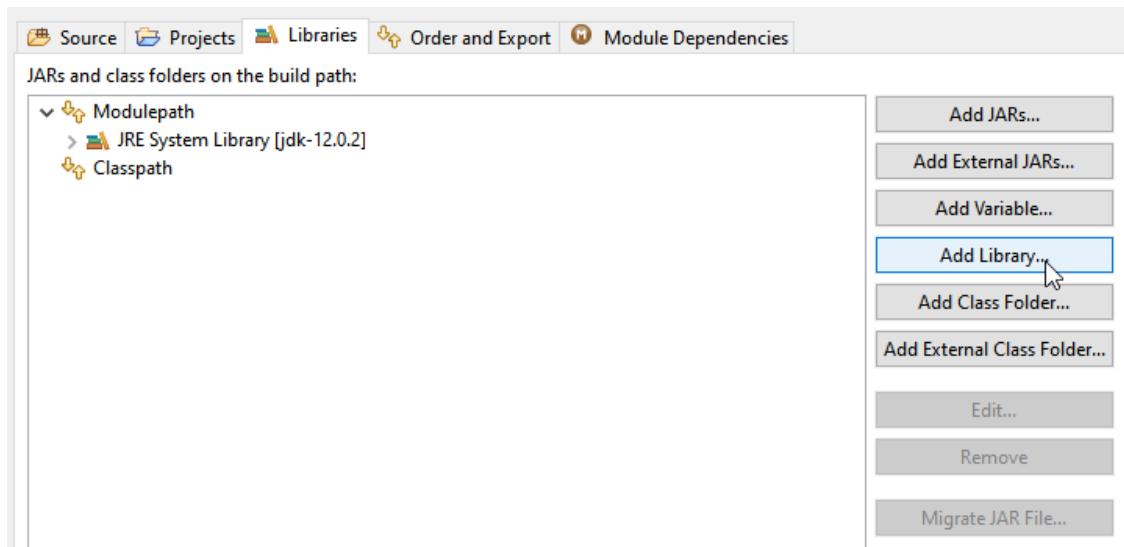
5. Ao abrir uma nova janela, selecione “Standard VM” e clique em “Next >”.



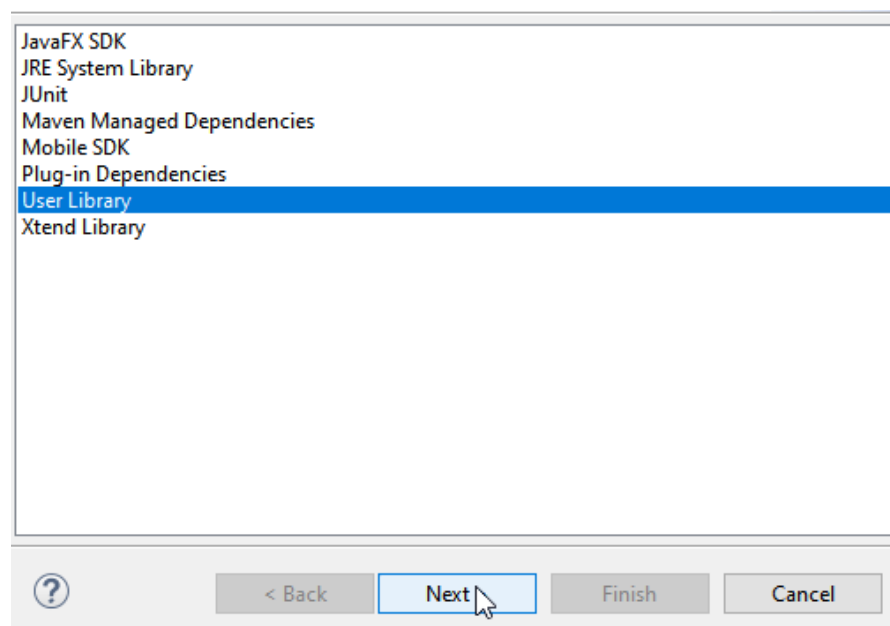
6. Uma nova janela abrirá para que seja selecionada a pasta com o JDK. Clique em “Directory” e escolha a pasta onde descompactou o arquivo .zip baixado no link do item 1.
7. Os outros espaços serão preenchidos automaticamente, então basta clicar em “Finish”.



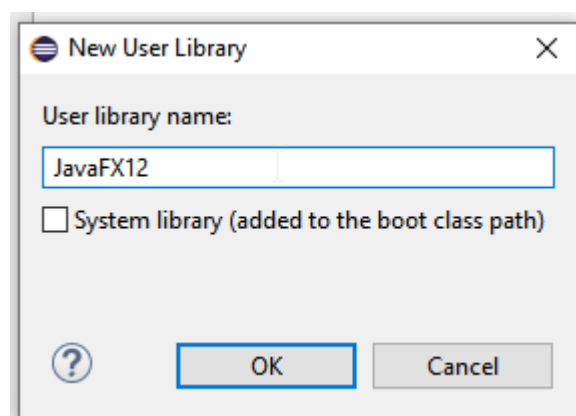
8. Ao retornar à janela de criação de projeto, no espaço “JRE”, selecione “use a Project specific JRE” e selecione o JDK configurado anteriormente.



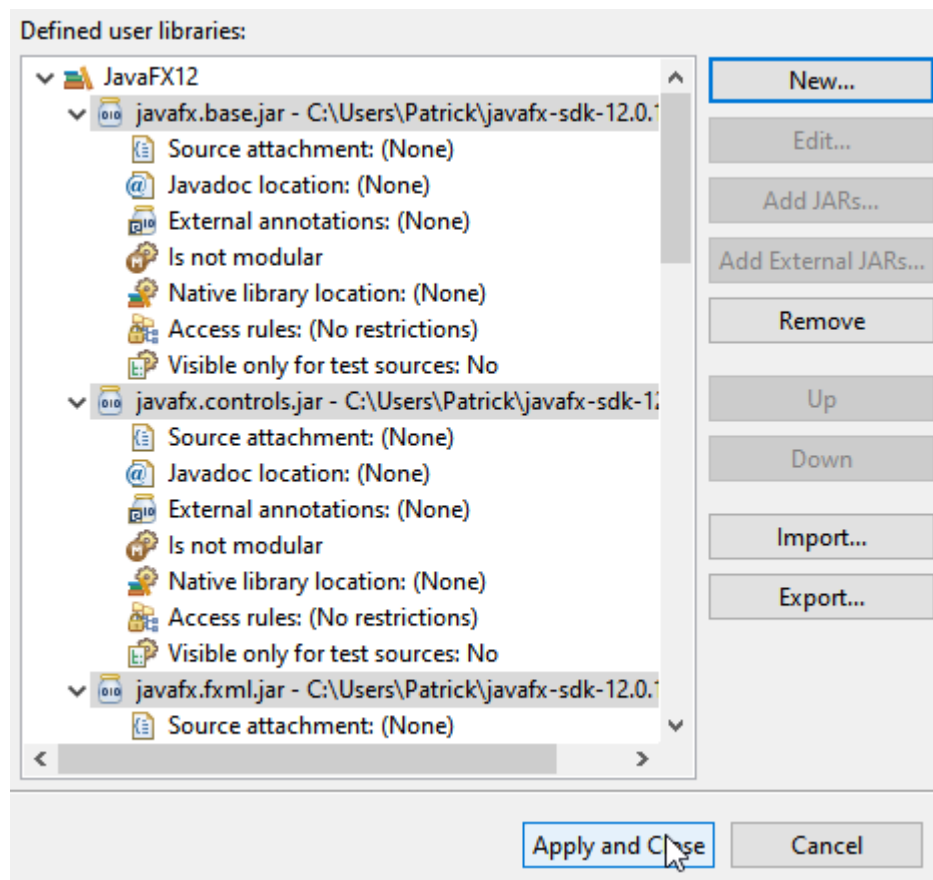
9. Especifique um nome ao projeto e clique em “Next”. Selecione então o espaço “Libraries”, clique em “Module Path” e em seguida em “Add Library”.



10. Na janela que abrirá, selecione “User Library” e clique em “Next”.



11. Uma nova janela abrirá. Clique no botão “User Library”, e digite o nome “JavaFX12” no espaço que aparecerá.



12. O nome digitado aparecerá na lista de bibliotecas. Selecione-o e clique em “Add External JARs...”. Então navegue até a pasta onde foi descompactado o pacote baixado pelo link do item 2. Procure, dentro dessa pasta, a pasta “lib”, contendo um conjunto de arquivos “.java”. Selecione todos e clique em “Open”. Aparecerão, então, os itens selecionados. Clique em “Apply and Close”.
13. Finalize a criação do projeto. Observe que no projeto deve aparecer uma pasta “JavaFX12” como biblioteca importada.
14. Copie para dentro do projeto a pasta “src” com os arquivos do programa. Selecione o arquivo “main” na pasta “application” e tente rodá-lo uma vez. Deve aparecer um erro.
15. Clique com o botão direito no projeto, selecione “Run As” e em seguida “Run Configuration”. Selecione a aba “Arguments” e no espaço “VM Arguments” adicione

```
--module-path "\path\to\javafx-sdk-12.0.1\lib" --add-modules  
javafx.controls,javafx.fxml,javafx.media
```

caso esteja no Windows, ou

```
--module-path /path/to/javafx-sdk-12.0.1/lib --add-modules ja-  
vafx.controls,javafx.fxml,javafx.media
```

caso esteja no Linux (tome cuidado para não copiar o texto com hifens). Substitua o “path\to” pelo endereço onde descompactou o pacote baixado no item 2. Para facilitar, pode copiar a pasta para “lib” para dentro do projeto, especificando no argumento da máquina virtual apenas o diretório “lib”. No Windows, ficaria

```
--module-path "lib" --add-modules  
javafx.controls,javafx.fxml,javafx.media
```

5 Implementação

Em uma visão ampla, o programa pode ser dividido em três esferas principais. Uma delas reúne as classes “*Leitor*” e “*Escritor*”, subclasses de “*GerenciadorPrincipal*”, responsáveis pela manipulação de arquivos no HD para o backup das informações do usuário. Nelas, são implementados os métodos de criação de pastas, leitura, escrita e busca de arquivos, todos estáticos, dado que são chamados utilizados em diferentes pontos do programa. Tais arquivos guardam instâncias das classes *Pomodoro* (e quaisquer uma de suas subclasses, i.e., *Atividade* e *ToDoList*) e *Perfil*. Para isso, as classes *Pomodoro* e *Perfil*, tais como quaisquer classes encapsuladas, devem implementar a interface “*Serializable*”, que permite com que a instância seja convertida em uma cadeia de números binários e escrita em um arquivo de extensão “*dat*”. Todas as classes que implementam a interface *Serializable* recebem um identificador, um tipo *Long* que é utilizado, na leitura de um arquivo, para verificar se a variável de referência que está recebendo aquele objeto possui o mesmo identificador, e, portanto, é compatível com o conteúdo do arquivo. As classes inerentes da linguagem java usualmente já implementam esta interface.

Os métodos são chamados ao longo do programa conforme o usuário cria ou edita atividades e perfis. A manipulação de arquivos pode acarretar *Exceptions* de Input/Output caso os arquivos estejam corrompidos ou tenham sido deletados, ou ainda se os objetos contidos nos arquivos não são compatíveis com a classe da

variável receptora (*ClassNotFoundException*); isso pode ocorrer quando o código das classes foi modificado, e um arquivo contém um objeto de uma versão anterior, ou se ocorre a tentativa de ler um arquivo contendo um objeto de uma classe diferente da variável receptora. Também é possível que um *NullPointerException* ocorra quando há algum erro na especificação do endereço de busca do arquivo, por exemplo, se o usuário passa um nome incorreto ou não especifica um nome de atividade ou perfil para ser buscado.

Na medida do possível, essas exceções não são passadas para as esferas superiores do programa, mas são resolvidas no nível dessas classes de gerenciamento, utilizando *“try with resources”*, pois utiliza-se a classe *“ObjectOutputStream”* para leitura dos arquivos, que implementa a função *“auto-closeable”*. Dessa forma, estes métodos possuem comportamento previsível, e se torna mais fácil desenvolver as outras partes do programa. Na ocorrência de um problema, os métodos retornam um endereço *“null”*, como meio de informar que a leitura ou escrita não foi feita. Essa informação é importante para programar na interface gráfica avisos de que o usuário está cometendo algum erro, como inserindo um endereço para um arquivo de extensão incompatível para o alarme.

Este método de backup pode ser ineficiente caso se trabalhe com um volume muito grande de informação, pois para cada edição, é preciso ler um objeto inteiro de uma só vez. Como troca, tem-se a facilidade de implementação. Uma alternativa seria utilizar arquivos de texto para guardar as informações de cada classe, entretanto, seria necessário desenvolver um sistema de leitura desses arquivos, o que é desnecessariamente trabalhoso.

Como é necessário manipular classes de grupos distintos, i.e., Perfil e Pomodoro, os métodos de leitura e escrita de objetos foram sobrecarregados, para preservar a simplicidade do programa, mantendo nomes semelhantes em métodos com mesma funcionalidade. Não é necessário, entretanto, criar um método para cada subclasse de Pomodoro por utilizarmos polimorfismo. Ao especificar um tipo “Pomodoro” como argumento de um método, pode-se passar quaisquer uma de suas subclasses.

Uma segunda camada do programa fica responsável pela manipulação dos dados. Nesta parte, consideramos as classes *“Atividade”* e *“ToDoList”*, que estendem a classe abstrata *Pomodoro*, e as listas duplamente encadeadas *“ListaPreparacao”* e

“ListaExecucao”, que estendem a classe abstrata *“ListaDePomodoros”*. Além disso, tem-se a classe *Perfil*.

Um pomodoro é meramente um período de tempo em que se faz algo, seguido de um intervalo de descanso. Dessa forma, não há realmente uma especificidade sobre o que deve ser feito, ou como deve ser feito, portanto, esta é uma classe abstrata que contém informações gerais de qualquer pomodoro, quais sejam, os tempos de execução, pausa, e os endereços para os arquivos de áudio que devem ser executados no início e no fim do pomodoro. Informações tais como “título” e “descrição” são atributos das subclasses. Os construtores de cada subclasse instanciam o objeto de forma hierárquica, fazendo a atribuição dos valores pertencentes ao seu nível, e passando para os construtores das superclasses os seus devidos parâmetros. A classe Pomodoro possui um construtor genérico para dar liberdade para a criação de subclasses que seguem um processo de instanciação distinto. A classe ToDoList se encaixa neste caso; consiste em uma lista de $n \geq 1$ tarefas, cada uma com seu próprio tempo de duração, assim, o tempo total da lista é a soma dos tempos de cada tarefa, portanto, na instanciação de uma ToDoList não é passado o tempo de duração e pausa como argumento, como tipicamente exige o construtor do Pomodoro.

Os construtores de cada uma dessas classes são sobrecarregados para se adaptar ao desejo do usuário de escolher ou não arquivos de áudio para uma atividade. Caso não queira, o construtor que não tem os endereços de arquivos como parâmetros fazem uma atribuição padronizada.

Uma vez que cada atividade possui a sua particularidade sobre como será executada, cria-se uma interface chamada *“GerenciaTimer”*, implementada pela classe Pomodoro e suas subclasses, com métodos que ditam o comportamento do contador. Dessa forma, utiliza-se polimorfismo para gerenciar o comportamento do programa; quando uma atividade da lista (de objetos Pomodoro) é iniciada, faz-se a chamada do método de interface, implementado de modo particular em cada tipo de atividade. Isso será retomado mais adiante.

A classe abstrata ListaDePomodoros também possui o propósito de reunir características e métodos gerais de uma lista de atividades, como inserção, remoção, obtenção de elementos no final e no início da lista. Seus elementos são objetos do tipo *“LinkedNode”*, que por sua vez encapsulam objetos tipo Pomodoro. Dessa forma,

pode-se utilizar essa estrutura para armazenar quaisquer subclasses de um Pomodoro. A subclasse *“ListaAtividades”* possui métodos de remoção e busca pelo título de uma atividade, e também implementa uma interface denominada *“GerenciaBackupAtividade”*, com métodos que utilizam as classes de gerenciamento de arquivo para salvar alterações nas atividades do usuário, ou removê-las.

É feita uma bifurcação com os outros dois tipos de listas, *“ListaPreparacao”* e *“ListaExecucao”*, pois estas cumprem propósitos distintos. A lista de atividades é um elemento de controle das atividades do usuário, e a lista de preparação, por sua vez, armazena uma lista de atividades que o usuário está selecionando para iniciar o timer. Antes de iniciar, ele pode alterar a ordem de execução, que será feita do último elemento ao primeiro (simulando uma fila de execução), tal como remover elementos selecionados. Para isso, é implementada uma interface chamada *“listaRealocacao”*, com os métodos para remoção de uma atividade da lista, e de realocação de uma atividade selecionada para uma posição acima ou uma abaixo. Essa realocação respeita regras distintas no momento de preparação da lista de execução e durante a execução em si, de modo que a lista de execução também implementa esta interface, mas de modo distinto. Quando se inicia o timer, a atividade que está sendo realizada não pode ser movida (mas pode ser cancelada), de modo que apenas as atividades da antepenúltima posição para trás podem ser realocadas. Essa restrição não existe durante a preparação.

Não foi utilizada uma estrutura de fila de prioridades, apesar do processo de execução de atividades seguir o comportamento de uma, para dar essa liberdade de reordenar os elementos.

Além de servir como uma espécie de contrato a ser respeitado pelas classes que as implementam, essas interfaces cumprem um importante papel no design do programa, separando claramente o que é um atributo ou um método particular de uma classe, e o que é um método que a classe deve utilizar para o funcionamento do programa. A realocação das atividades, por exemplo, é feita tendo em mente como o usuário irá interagir com o programa e quais as regras dessa interação, e não como um tipo de lista deve se comportar, enquanto objeto.

Enfim, a classe Perfil contém as informações do usuário. Atualmente, contém apenas seu nome, e uma lista de atividades, mas pode-se pensar em adicionar

estatísticas de uso, como número de pomodoros executados por data. A existência da Lista de Atividades é vinculada ao perfil, i.e., a classe *ListaAtividades* é encapsulada pela classe *Perfil*, pois ao se fazer login, é gerada uma instância de um *Perfil*, e é feita automaticamente a chamada do construtor da *ListaAtividades*, passando como argumento o nome do perfil. Esse construtor fará a leitura de todos os arquivos de atividades vinculado ao perfil, utilizando as classes de leitura de arquivos, e qualquer modificação será salva utilizando o nome do perfil para encontrar os diretórios adequados.

Neste ponto, na ocorrência de uma exceção, a atividade da qual decorreu o erro de leitura é apenas ignorada.

A classe *Perfil* também implementa um método de backup, chamado *“GerenciaBackupPerfil”*. Esta interface cumpre o mesmo papel da interface para backup de atividades, entretanto, seus métodos (de leitura e remoção) não recebem parâmetros, dado que a chamada é feita internamente na classe *Perfil*, passando a si mesma como parâmetro para a chamada dos métodos estáticos de manuseio de arquivos.

O endereço do objeto dessa classe (*Perfil*) é passado para as classes de controle das interfaces gráficas sempre que uma janela é chamada como uma forma de vinculá-las ao usuário, e estabelecendo um fluxo de informação. Esse mecanismo é utilizado, por exemplo, na janela de apresentação das atividades do usuário, em que um método denominado *“initialize”* é executado no instante em que a janela é aberta, análogo ao método *“main”* da classe principal de um programa, recebendo o perfil como parâmetro, e se utilizando dele para buscar as informações do usuário. As janelas de edição de atividades também utilizam este objeto para salvar os arquivos editados.

A última esfera do programa diz respeito à interface gráfica. A interface gráfica (IU – Interface de Usuário) foi desenvolvida utilizando o pacote JavaFX. Neste pacote, cada elemento da interface, como um campo de texto ou um botão, é uma instância de uma classe, com seus atributos e métodos particulares. Os elementos gráficos (layout) dessas classes são especificados em um arquivo particular com extensão *“fxml”*, e cada uma delas são encapsuladas em uma classe central, usualmente denominada de Controlador (Controller), responsável por gerenciar o comportamento

da IU, especificando como esses elementos se comportam, interagem uns com os outros e com as outras esferas do programa (backup de arquivos e processamento de informação).

O uso de interfaces gráficas configura uma perspectiva de desenvolvimento de sistemas distinta da perspectiva tipicamente procedural, em que o programa executa uma série de procedimentos do início ao fim. O programa passa a se basear em eventos, em que cada método é ativado por meio da interação do usuário com um dos elementos da interface. Nesse sentido, na classe controladora são implementados os métodos vinculados a esses eventos, como o apertar de um botão, a seleção de um elemento com o mouse, o direcionamento de outputs ou o resgate de inputs do usuário.

O programa possui 5 janelas: uma janela de seleção do perfil do usuário, uma em que são apresentadas as atividades do usuário, podendo criar, remover e editar atividades, e escolher atividades para a ativação do timer, uma janela para a edição/criação de um objeto Atividade e uma para um objeto ToDoList e, por fim, uma janela para o timer. Cada uma dessas janelas possui uma classe controladora.

A implementação de cada uma dessas janelas será apresentada seguindo a lógica de uso do programa.

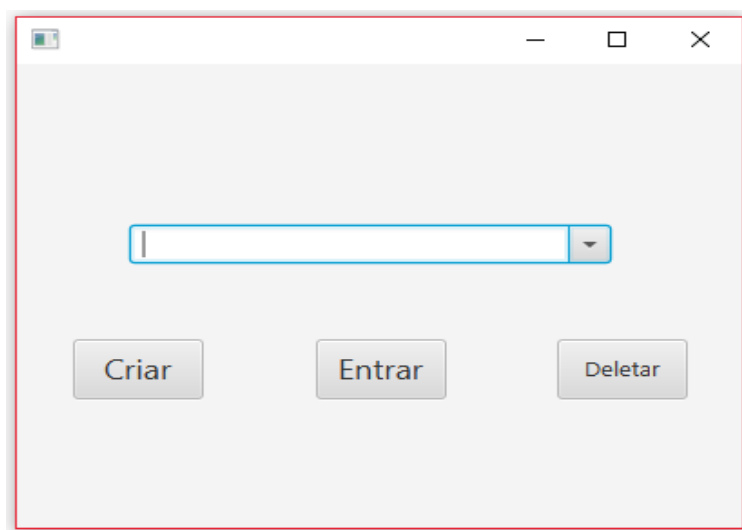


Figura 5 Janela de seleção de um perfil.

1. Janela de Login

A primeira janela (Figura 5) é um sistema de login, gerenciado pela classe *"controladorJanelaPerfil"*. Ao abrir, o método *"initialize"* é executado automaticamente, fazendo uma busca no diretório de backup por arquivos de perfis já criados, e

alimentando a lista de seleção com seus nomes. O usuário pode editar o campo de texto da lista, a fim de criar um novo perfil, ou selecionar algum para remoção ou login. Os métodos de criação e remoção são executados apenas se o campo de texto não está vazio, e as exceções (erros de input/output) são tratadas no nível dos métodos de gerenciamento de arquivos. Uma tentativa de login com um perfil inexistente pode acarretar um `IOException` ou `NullPointerException` na chamada da janela seguinte, que tentará carregar os dados do perfil. Nesse caso, um bloco try-catch no método de inicialização da janela seguinte impede a sua abertura, retornando para a tela de login.

2. Janela De Atividades

A tela seguinte (Figura 3) carrega automaticamente as atividades do usuário na sua inicialização, apresentando-as em uma lista (um objeto tipo `ListView` – lista à esquerda). Essa lista permite interação com o mouse. Clicando em uma atividade, suas informações serão apresentadas na tela, e clicando duas vezes, a atividade é adicionada na lista de preparação (à direita). É feito um tratamento de exceções para que o método que governa a interação com o mouse não acarrete erros ao tentar acessar uma lista vazia, dado que ele é executado como resposta ao evento de interação com o mouse.

No menu superior, em “Arquivo”, o usuário pode escolher as opções de remover ou editar um item selecionado, criar uma nova Atividade ou uma `ToDo List`, ou ainda deslogar, voltando para a tela anterior. Os métodos para remover ou editar uma atividade também passam por um tratamento de exceções, dado que eles também tentam acessar o objeto `ListView` da interface para obter a atividade selecionada, e isso pode acarretar um `NullPointerException` ou `IOException`, se a lista estiver vazia ou se o usuário deliberadamente tentou selecionar um espaço em branco.

3. Janela de Criação/Edição de uma Atividade

Ao selecionar no menu a opção de criar uma atividade, a janela de edição (Figura 1) é aberta.

A janela anterior permanece aberta, mas inacessível até que se feche a janela de edição. Se o usuário selecionar o objeto “*CheckBox*”, especificando que deseja customizar o alarme, os elementos de busca são habilitados, e clicando no botão “buscar” uma nova janela é aberta para que o usuário especifique um arquivo de áudio. Clicando em “OK”, um objeto Atividade é instanciado e salvo no diretório de backup

do usuário, chamando o construtor adequado em função da decisão do usuário de customizar ou não um alarme.

A extensão do arquivo escolhido para o alarme é verificada antes da instanciação, mas até o momento apenas uma mensagem é lançada no terminal (quando o programa é executado pelo Eclipse), e a criação é interrompida, obrigando o usuário a escolher um arquivo correto. Um tratamento de exceção é feito, entretanto, durante a execução do timer, quando há a tentativa de tocar um arquivo que pode estar corrompido, que seja inexistente ou que não seja de nenhum formato de áudio permitido.

4. Janela de Criação/Edição de uma ToDo List

A dinâmica de criação de uma Lista de Tarefas é essencialmente a mesma, mas respeita as regras de criação deste objeto, e admite uma janela de criação própria (Figura 2). Um objeto `ToDoList` pode ser instanciado passando o inteiro “0” como parâmetro para o tamanho do seu vetor de tarefas, dado que a lista na interface para a seleção deste número, um objeto “ComboBox”, permite edição pelo usuário. Entretanto, para habilitar os itens de criação de tarefas na interface, o usuário deve selecionar na lista “N. Tarefas” um inteiro $n \geq 1$. Ao tentar adicionar novas tarefas, o método de inserção da classe `ToDoList` é chamado, e nela é feita a verificação do número de elementos no vetor, retornando o boolean “false” no caso em que o vetor está cheio. Portanto, o usuário fica restrito a adicionar um número menor ou igual ao limite de tarefas especificado.

Clicando no item “Editar” do menu na janela de atividades, as mesmas interfaces de criação são abertas, mas carregando as informações da atividade selecionada. Possibilitando a alteração das descrições e dos alarmes.

Aqui, há um bug em que, caso o usuário tente editar o título de uma atividade, é criada uma nova, e não é feita a edição da anterior. Isso se dá porque os métodos de escrita de arquivos trabalha com objetos inteiros, então na edição, cria-se um novo objeto com o novo nome, que é salvo na pasta do usuário. Na edição de uma lista de tarefas, caso o usuário tente modificar o seu limite, é feita uma nova instanciação, e caso o usuário tente confirmar a edição, o objeto é sobrescrito, perdendo as tarefas anteriores.

Uma correção possível para o primeiro problema consiste em fazer uma remoção da atividade anterior antes de escrever a nova atividade editada, a fim de garantir a “troca”. No caso da lista de tarefas, deve-se escrever métodos internos na classe `ToDoList` para fazer modificações no tamanho do array de tarefas, preservando as atividades já adicionadas.

5. Janela do Timer.

Por fim, ao clicar em “Iniciar Timer”, a janela do contador é aberta, carregando as atividades inseridas na Lista de Preparação, e iniciando o timer automaticamente, que fará a contagem regressiva das durações e das pausas de cada atividade, da última à primeira, por meio de um loop que verifica o número de elementos na lista de execução e remove as atividades finalizadas.

O gerenciamento do contador é feito pela própria instância da atividade, que implementa uma interface com este propósito. Neste ponto, o recurso de polimorfismo se mostra bastante útil. No nível da interface gráfica, o controlador apenas chama, dentro do loop, um método de um objeto `Pomodoro` genérico, implementado de maneira particular pelas subclasses (`ToDoList` e `Atividade`).

Como estes métodos fazem modificações na interface gráfica, é necessário passar como argumento as referências dos seus objetos relevantes.

Para a implementação do contador, utilizou-se o recurso de “`MultiThreading`”, que permite processamento em paralelo. Utiliza-se a classe “`Thread`”, e ao instanciar um objeto desta classe e utilizar seu método “`start`”, o método dentro do qual a “`thread`” foi iniciada passa a executar em paralelo com o restante do programa. Dessa forma, pode-se, em uma “`thread`”, executar os processos relativos ao contador de tempo, e na outra `thread` principal ocorre o processamento das interações do usuário com a interface gráfica.

Ao fim da contagem regressiva, deve-se tocar um áudio do alarme escolhido pelo usuário (ou o padrão). Neste ponto, pode ocorrer exceções tipo `MediaException` e `NullPointerException`, caso haja erro de leitura dos arquivos. É necessário utilizar o método “`getClass().getResource().toString()`” para gerar uma URL que será passada para a classe “`AudioClip`”, do `JavaFX`, por meio da qual se executa o arquivo. Entretanto, tivemos problemas para fazer o método “`getResource()`” retornar o endereço do arquivo, funcionando apenas quando o arquivo era colocado dentro da

pasta do programa. O programa não deixa de executar com essas exceções, entretanto.

6 Conclusões

As funcionalidades básicas da proposta do projeto foram implementadas. Não houve tempo hábil, entretanto, para a implementação de um sistema de estatísticas do usuário e um desenvolvimento mais detalhado das possibilidades do uso de polimorfismo para o gerenciamento do timer, uma vez que isso envolve, também, o design de novos tipos de atividades e interfaces gráficas para sua manipulação.

Os conceitos de Orientação a Objetos se mostraram fundamentais durante a etapa de design do programa. A complexidade do sistema é reduzida quando se separa seus elementos em um conjunto de objetos com atributos e comportamentos próprios, tal como mecanismos de interação entre objetos. A fase mais longa de desenvolvimento consistiu precisamente em estabelecer o que o programa deve fazer, como se dará a interação com o usuário, e quais objetos e métodos serão necessários para obter o comportamento desejado.

Algumas decisões de design são questionáveis, do ponto de vista de implementação, como a utilização de interfaces em algumas partes do programa - seria possível implementar os mesmos mecanismos como métodos de classe e atingir o mesmo resultado. Entretanto, o uso de interfaces tem uma utilidade do ponto de vista do design, que é o de separar um comportamento próprio de um objeto com o modo como ele deve interagir com outros objetos.

De modo análogo, as classes de gerenciamento de backup poderiam ser implementadas em uma única classe, contemplando métodos de escrita e leitura. Por outro lado, a separação das classes determina responsabilidades específicas para elementos do programa, então compreende-se que há uma classe responsável apenas por leitura, e outra apenas por escrita, herdando da superclasse métodos gerais para navegar por diretórios, criar ou remover arquivos.

A hierarquia de atividades poderia ser estendida, contemplando um tipo mais simples de Pomodoro que possuiria apenas atributos de tempo de duração e pausa. Essa generalidade possibilitaria um leque maior de customização das subclasses, mas pode ser, também, uma complicação desnecessária do sistema, adicionando mais itens do que o necessário.

Uma grande dificuldade consiste em elaborar como se dará a interação do usuário com o programa, o que ele pode ou não fazer e que tipo de interação acarretaria erros. O tratamento desses erros pode ser elaborado no nível do código, com tratamento de exceções e condicionais para verificação, ou por decisões de design, delimitando o que pode ou não ser feito, por exemplo, ao fazer um condicional simples para verificar se foi digitado uma string incorreta para o atributo que se pretende instanciar. Um outro exemplo está na escolha dos tempos de pausa e duração. Decidiu-se utilizar listas de números a serem escolhidos e não oferecer um espaço de texto simples para que o usuário digitasse o tempo desejado, pois é mais simples controlar o input do usuário, fazendo menos verificações, dentro do código, do que ele está tentando passar como argumento.

7 Referências Bibliográficas

- Davies, D. R., & Parasuraman, R. (1982). The psychology of vigilance. London: Academic Press.
- Ariga, A., & Lleras, A. (2011). Brief and rare mental “breaks” keep you focused: Deactivation and reactivation of task goals preempt vigilance decrements. *Cognition*, 118(3), 439-443.