E-fólio B | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Programação por Objetos

CÓDIGO: 21093

DOCENTES: Jorge Morais, Leonel Morgado e Rúdi Gualter (tutor)

A preencher pelo estudante

NOME: Henrique Alexandre Cortez Melo

N.º DE ESTUDANTE: 2201211

CURSO: Licenciatura em Engenharia Informatica

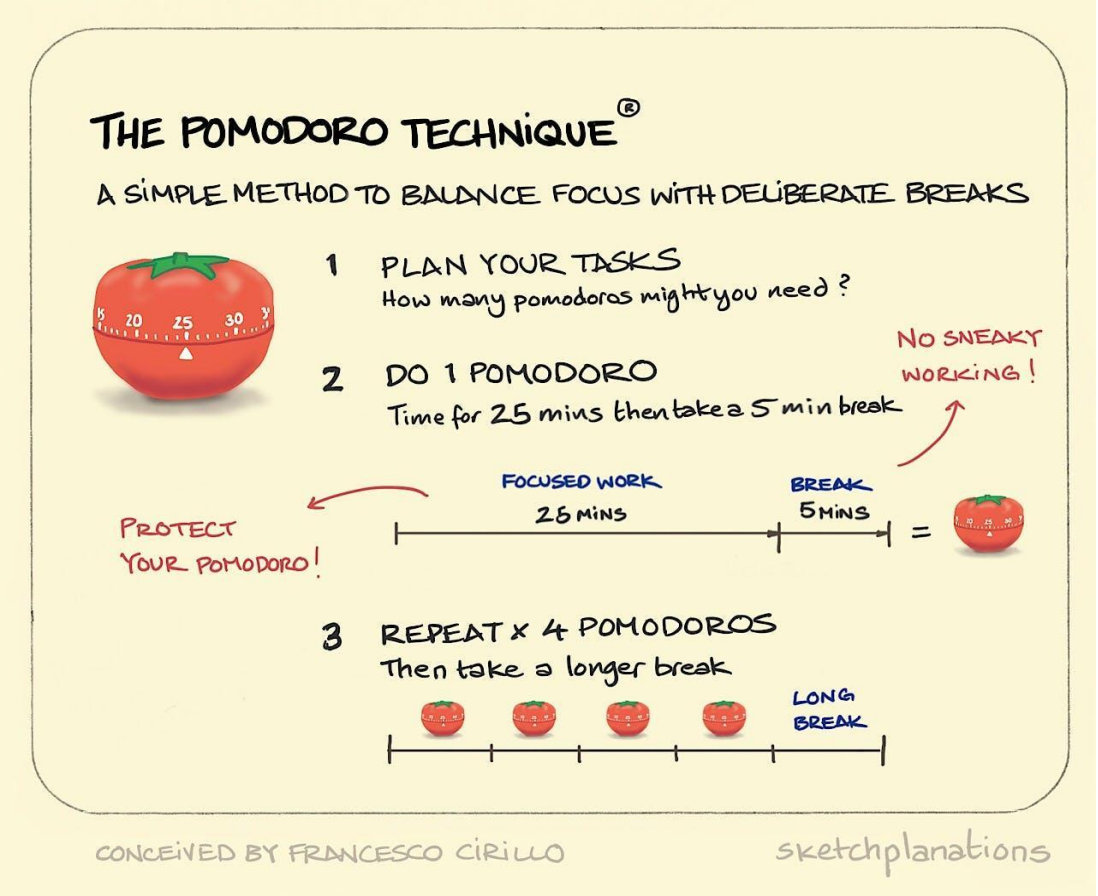
DATA DE ENTREGA: 26/12/2023

TRABALHO / RESOLUÇÃO:

Como projeto escolhido para esta disciplina foi eleita uma pequena ferramenta de autoajuda para o estudo. Esta ferramenta segue o método "**Pomodoro**". Resumidamente, a ferramenta terá um contador decrescente, mostrando assim ao utilizador, quanto tempo "produtivo" existe restante. Assim que o contador chega a 00:00 deverá imediatamente mostrar ao utilizador o tempo de pausa restante, marcando também que uma sessão de estudo foi executada.

Explicação resumida do método "**Pomodoro**" :

https://todoist.com/productivity-methods/pomodoro-technique



O que faltava fazer (De acordo com EfolioA):

1. Ter a opção de pausar o contador. ✔

- Para alcançar este objetivo terá sido feito um novo método, **“pause_countdown”,** este método ao ser chamado atualiza a interface, parando assim o contador se este estiver a decorrer, e continuado o contador se este estiver já pausado. Para o auxílio deste método foi criada uma variável booleana em **PomodoroTimer**, titulada de **pause_pressed.** Adicionalmente terá sido adicionado um botão na janela principal da Interface, titulado de **pause_button.**

2. Ser notificado na mudança de estados, a partir de um pequeno alarme e/ou som. (Deixado para efolio Final)

3. Ter a opção de modificar a duração de ambos os tempos de estudo e repouso.✔
   - Para isto foi criado o método **change_durations,** este método cria uma janela com quatro botões, três campos de formulário e três rótulos informativos, permitindo ao utilizador a inserção de uma nova duração(em minutos) para todos os três períodos(descanso, trabalho e descanso longo), para evitar possíveis bugs, foi decidido desativar a janela principal, enquanto a janela de mudar durações estiver ativa. Adicionalmente criou-se o método **close_window**, fechando este a janela de durações(atribuído ao **close_window_button**).

4. Correção de possíveis erros e/ou bugs. ✔
   - Foi detetado um erro, onde se o utilizador premisse várias vezes o **start_button,** o contador saltava entre os vários períodos de trabalho, descanso e descanso longo em cada segundo, para impedir que esta situação seja replicada, os métodos foram ajustados para desativar e reativar o **start_button** quando necessário. Garantindo assim que o contador siga uma lógica mais controlada e por este meio evitando comportamentos inesperados.

Implementação de conceitos de programação por objetos pretendidos:

Herança e Subclasses:

Devido à natureza do projeto em si, não se terá criado uma classe mãe que pudesse ser utilizada e recriada em várias situações diferentes. Visto que no projeto terá sido apenas necessário uma instância de cada classe. No entanto, de maneira a exemplificar e demonstrar o conceito de subclasses e heranças, para cada classe terá sido feito um “Template”, ou seja, um modelo exemplar.

Estes dois modelos(**Timer_Template** e **UI_Template**) iniciam as variáveis aplicadas às suas subclasses, e adicionalmente, possuem métodos abstratos que servirão como ponto de entrada, ou seja, as subclasses precisaram de implementar os respetivos métodos já presentes nos seus “templates”.

Esta abordagem permite definir uma estrutura comum, garantindo que certos comportamentos essenciais sejam implementados de maneira consistente quando criando subclasses.

Sobrecarga de Métodos:

Embora não seja sobrecarga de métodos tradicional, tendo estas normalmente assinaturas múltiplas, o método **save_duration** no código demonstra uma forma de sobrecarga.

```
def save_duration(self, form, global_var):
    new_duration = form.get()
    try:
        new_duration = int(new_duration)
        if global_var is self.my_timer.WORK_MIN:
            self.my_timer.WORK_MIN = new_duration
        elif global_var is self.my_timer.SHORT_BREAK_MIN:
            self.my_timer.SHORT_BREAK_MIN = new_duration
        elif global_var is self.my_timer.LONG_BREAK_MIN:
            self.my_timer.LONG_BREAK_MIN = new_duration
    except ValueError:
        # If not valid Integer
        messagebox.showerror(message="Invalid input. Please
```

```
enter a valid integer")
```

Sendo este o único método que lida com diferentes tipos de durações (duração de trabalho, duração de descanso, duração de descanso longo) e assim aceitando diferentes parâmetros.
Demonstrando assim uma abordagem flexível para lidar com vários casos num único método, alcançando um resultado muito semelhante à sobrecarga de métodos.

Polimorfismo:

O método **count_down** utiliza polimorfismo ao trabalhar de forma dinâmica com diferentes tipos de dados (números inteiros e strings) durante a formatação do texto do timer.

```
def count_down(self, count):

    # Conversão de minutos e segundos
    count_min = floor(count / 60)
    count_sec = count % 60

    if count_sec == 0:
        count_sec = "00"
    elif count_sec < 10:
        count_sec = f"0{count_sec}"

    if count_min == 0:
        count_min = "00"
    elif count_min < 10:
        count_min = f"0{count_min}"

    if not self.pause_pressed:
        self.ui.canvas.itemconfig(self.ui.timer_text,
text=f"{count_min}:{count_sec}")
        if count > 0:
            self.my_timer = self.ui.window.after(1000,
self.count_down, count - 1)
        else:
            self.start_timer()
            mark = ""
```

```
            sessions = floor(self.REPS / 2)
            for rep in range(sessions):
                mark += "🍅"
            self.ui.checkmark_label.config(text=mark)
    else:
        self.my_timer = self.ui.window.after(1000,
self.count_down, count)

# Modificar o texto de acordo com o pretendido
def start_timer(self):
    self.ui.start_button.config(state=tkinter.DISABLED)
    if self.REPS % 2 == 0:
        self.ui.timer_label.config(text="STUDY MELO STUDY",
fg=self.ui.GREEN)
        self.count_down(self.WORK_MIN * 60)
    elif self.REPS == 8:
        self.count_down(self.LONG_BREAK_MIN * 60)
        self.ui.timer_label.config(text="REST MELO REST",
fg=self.ui.PINK)
    else:
        self.count_down(self.SHORT_BREAK_MIN * 60)
        self.ui.timer_label.config(text="REST MELO REST",
fg=self.ui.PINK)
    self.REPS += 1
```

Isto acontece com a conversão de minutos e segundos em strings, permitindo que o método manipule ambos os tipos sem a necessidade de verificações explícitas.
Seguindo assim o princípio do polimorfismo, onde diferentes tipos são tratados de forma uniforme por via de uma interface comum.

Abstração:
Abstração envolve simplificar sistemas complexos, encapsulando detalhes e expondo apenas características essenciais. A classe **PomodoroUI** abstrai os elementos da interface do utilizador e as interações do contador. Os utilizadores interagem com o contador por meio de métodos bem definidos, como **start_timer**, **reset_timer**, **pause_countdown** e **change_durations**.

Este encapsular oculta os detalhes de implementação subjacentes,

proporcionando uma interface clara e simplificada para os utilizadores interagirem com o temporizador, demonstrando o princípio da abstração.

Resumidamente, o código exibido demonstra uma forma de sobrecarga de métodos, demonstra polimorfismo por meio da tipagem dinâmica e destaca a abstração ao encapsular as complexidades do contador dentro de classes e métodos bem definidos.

Embora a sobrecarga de operadores não seja explicitamente demonstrada neste código, os outros conceitos destacam a flexibilidade e versatilidade do Python ao lidar com os diferentes dilemas da PPO.

O que falta fazer:

- Ser notificado na mudança de estados, a partir de um pequeno alarme e/ou som.
- Mudança de tema, mudando assim as cores do programa (Caso tempo permita, feature opcional/extra).