Kusztelak Paulina

Okołowicz Bernadeta

Paradygmaty programowania

APLIKACJA DO ZARZĄDZANIA KSIĄŻKAMI - SPRAWOZDANIE

Spis treści

Część I. Opis projektu

Zadaniem do wykonania było wykorzystanie własnego pomysłu i przedstawienie go w postaci kodu zgodnego z zasadami programowania obiektowego oraz GUI (Grafical User Interface).

W oparciu o prywatne doświadczenia, zdecydowałyśmy się na stworzenie aplikacji, dzięki której możemy zarządzać swoimi książkami. Aplikacja została stworzona na zasadzie szkieletu, który można w późniejszym etapie rozwijać o dodatkowe opcje np. filmy.

Naszym zamysłem była aplikacja, która nie wymagałaby zbyt dużego zaangażowania ze strony użytkownika i była dość prosta w obsłudze. Miałaby za zadanie pobierać od użytkownika dane dotyczące książek lub filmów, zgromadzonych w postaci dokumenty tekstowego, w którym każdy tytuł zaczyna się od nowego wiersza. Z poziomu interfejsu użytkownik mógłby swobodnie zarządzać tytułami, a w przypadku książek mógłby również zapisywać informacje, jaką książkę, komu i kiedy pożyczył lub jaką książkę od kogoś pożyczyliśmy. Można również usuwać dodane książki, jeśli się je zgubiło, zniszczyło lub oddało.

Projekt zaczęłyśmy od stworzenia programu zgodnego z zasadami programowania obiektowego jednak bez graficznej wizualizacji, aby dopracować kod na podstawowym poziomie. Kolejnym etapem było przekształcenie kodu poszczególnych funkcji w taki sposób, aby działały zgodnie z naszym zamysłem graficznej wizualizacji.

Część II. Opis kodu (bez wizualizacji)

Klasa App:

Jest ona ogólnym szkieletem zawierającym podstawowe informacje i opcje na temat nazwy, wersji oraz działania kodu. Na początku zdefiniowałyśmy zmienne przechowujące kolejno: informację o nazwie aplikacji, wersji, a także zmienne, które przechowują tekst wyświetlany użytkownikowi podczas jego wyboru oraz przy wyborze złej opcji. Zmienna „quit" służy nam do kończenia pracy aplikacji.

W konstruktorze funkcji zdefiniowałyśmy dwie puste listy, jedną dla książek dodanych – „self.books" i drugą dla książek załadowanych z pliku zapisanego w poprzedniej sesji użytkownika – „self.loaded_books" oraz dwa puste słowniki dla menu (dostępnych opcji) oraz akcji (czynności związanych z daną opcją), została tutaj też ustawiona flaga – „False", którą będziemy sprawdzać, czy została wybrana opcja przejścia do innych, dodatkowych opcji (sprawdzamy tu, czy należy przejść do nowej klasy), następnie do metody „opt(self, key, value, procedure)" przekazujemy kolejno informacje o trzech ogólnych opcjach do wyboru po uruchomieniu aplikacji – „Quit", „Info" oraz „More".

W metodzie „info(self)" zwracamy sformatowaną wiadomość o nazwie i wersji aplikacji, a także krótką wiadomość do czego służy.

Za pomocą metody „choice(self)" wyświetlamy użytkownikowi tekst składających się z elementów słownika menu, każdy element ma postać „[klucz]wartość", np. „[M]More", następnie pytamy użytkownika o to, którą opcję z dostępnych wybiera. Dla wygody użytkownika zamieniamy wprowadzone litery na wielkie (tzn. użytkownik wpisuje literę wybranej opcji w dowolnym formacie – małą lub wielką literą).

Metoda „opt(self, key, value, procedure)" dodaje nowy wpis do słownika „self.menu", gdzie kluczem jest „key", a wartością „value". Uzupełnia menu o nowe opcje oraz dodaje wpis do słownika „self.action", gdzie kluczem jest „key", a wartością „procedure". Ten krok uzupełnia listę procedur, które będą wykonywane w zależności od wybranej opcji.

Metoda „print_badchoice(self)" przekazuje użytkownikowi informację w momencie, gdy wybierze opcję, której nie ma w menu.

Metoda „run_app(self)" jest „siłą napędową" aplikacji, jej działanie opiera się na pętli, w której odpytujemy użytkownika o wybór opcji i przypisuje go do zmiennej „result", a następnie sprawdza, czy „result" jest różna od „quit" i nie ma wartości ustawionej wcześniej flagi – „self.more_chosen", jeśli tak, to status pozostaje „True" i pętla działa dalej, jeśli status ma wartość „False", metoda kończy działanie programu, zwracając „self.quit". Jeśli klucza nie ma w słowniku „self.actions" użytkownik otrzymuje informację o błędnym wyborze.

Za pomocą metody „quit(self)" kończymy pracę aplikacji, w metodzie „more(self)" zmieniamy flagę na „True", co oznacza, że opcja dodatkowa została wybrana i program powinien

przenieść nas do wybranej klasy. W tym celu przypisujemy instancję klasy „MainApp” i następnie przekazujemy ją do metody „navigate_to(self, target class)”, która jest odpowiedzialna za nawigację do innej klasy. Przekazuje ona klasę docelową jako argument.

Metoda „navigate_to(self,target_class)” ma za zadanie przenieść użytkownika do nowej klasy, za pomocą wybranej opcji. Przyjmuje ona klasę docelową jako argument - „new_instance = target_class(self)”. „If self.more_chosen /else target_class()” - Jest to wyrażenie warunkowe, które tworzy nową instancję klasy docelowej (target_class). Jeżeli flaga „more_chosen” jest ustawiona na „True”, używamy „self” (obecnej instancji klasy) jako argumentu dla konstruktora klasy docelowej, w przeciwnym razie, gdy flaga jest ustawiona na „False”, tworzymy instancję klasy bez przekazywania żadnych dodatkowych argumentów. Za pomocą „new_instance.run_app()” wywołujemy metodę „run_app()” na nowo utworzonej instancji klasy docelowej.

Klasa Main App:

Klasa ta dziedziczy po klasie „App”. W konstruktorze klasy zdefiniowałyśmy do wyboru opcje, czym dokładnie chcemy zarządzać – można wybrać zarządzanie książkami (Books) lub filmami (Movies).

Po wyborze odpowiedniej opcji, specjalnie zdefiniowane metody klasy za pomocą „try-except” ponownie zmieniamy flagę na „True”, przypisujemy instancję klasy, a następnie wywołujemy metodę „navigate_to(self, target_class)”.

Klasa BookShelf:

Klasa dziedziczy po klasie App. W konstruktorze zostały zdefiniowane kolejne opcje, dostępne tylko w przypadku wybrania opcji „Books”.

Po wyborze opcji „Add new books” zostaje wykonana metoda „add_books(self)”, która prosi użytkownika o podanie ścieżki do pliku, a następnie za pomocą klauzuli „try – except” otwiera ścieżkę w opcji „read” i iterując po jej wierszach pobieramy tytuł książki dodając go do słownika. Zakładamy tutaj jak najbardziej optymalne podejście do użytkownika, zatem w pliku z książkami uwzględniamy tylko ich tytuły, bez nazw autora. Ponieważ zarządzamy własnymi książkami, dodawanie autora uznałyśmy za zbędne i bardziej czasochłonne dla użytkownika. Następnie po dodaniu tytułów do słowników, dodajemy je wszystkie po kolei do listy. Jeśli słowniki zostały dodane pomyślnie użytkownik otrzymuje komunikat, że książki zostały dodane pomyślnie, a metoda zwraca listę, jeśli nie, to zostaje wyświetlony komunikat o niepowodzeniu. Jeśli wystąpiły błędy, zostają one również odpowiednio wyświetlone użytkownikowi.

Kiedy użytkownik wybierze opcję „Remove books” zostaje wykonana metoda „remove_books(self)”, w której dopytujemy użytkownika o tytuł książki, który chce usunąć, aby było to bardziej przyjazne, została tutaj dodana metoda „.capitalize()” , która sprawia, że wprowadzony tytuł rozpoczyna się od wielkiej litery. Po podaniu tytułu została zdefiniowana zmienna „book_found”, która ustawiona jest na „False” i po niej rozpoczyna się iteracja najpierw po wartościach listy, a następnie po wartościach i kluczach słowników. Jeśli w danym słowniku w tytule znajduje się podana przez użytkownika książka, usuwamy cały słownik z tym tytułem, a zmienna „book_found” zamienia się na wartość „True” i zostaje zwrócona lista z książkami. Zostało tutaj również wykonane sprawdzenie tytułu książki, w książkach załadowanych z zapisanego przez nas pliku (było to konieczne, gdyż ten plik jest w formacie „.json”, a nie „.txt”), algorytm działania jest taki sam.

Opcja „Show books” sprawdza na początku, czy lista ze słownikami książek nie jest pusta, jeśli tak, to przechodzi do iteracji po jej wartościach i następnie tworzymy zmienną „book_info”, która oznacza utworzenie listy, gdzie każdy element tej listy to string zawierający informacje o konkretnej książce. Wykorzystuje ona tzw. „list comprehension”, który jest skróconą i bardziej zwięzłą formą zapisu pętli. Następnie wyświetlamy tą listę, w której każdy zespół „klucz:wartość” oddzielony jest znakiem „|”. Ten sam algorytm działania stosujemy w przypadku książek załadowanych z pliku zapisanego przez użytkownika w poprzedniej sesji. Jeśli obie listy są puste, wyświetlany jest komunikat o braku książek.

Za pomocą opcji „Categorize” możemy wybrać kolejne opcje, takie jak:

- „Borrow books”, po której wykonywana jest metoda „borrowed_books(self)”. W tej metodzie użytkownik proszony jest o podanie tytułu książki pożyczonej, imienia osoby, od której książka została pożyczona oraz daty pożyczenia książki. Imię i tytuł książki zapisywane są od wielkiej litery. Pozyskane informacje zapisywane są w słowniku „borrowed_book”, odpowiednie zmienne są przypisywane jako wartości do przyporządkowanych kluczy. Następnie sprawdzamy, czy lista książek dodanych lub załadowanych z pliku nie jest pusta, jeśli nie, to do listy dodajemy słownik „borrowed_book” z odpowiednimi informacjami i zostaje wyświetlony komunikat o pomyślnym zakończeniu operacji. Jeśli książki nie zostały dodane wyświetla się komunikat o braku książek.

- „Return books”, po której wykonywana jest metoda „return_book(self)”. Na początku prosimy użytkownika o podanie tytułu książki zwróconej i zapisujemy go w zmiennej „n_book”, tytuł automatycznie zaczyna się od wielkiej litery, następnie ustalamy zmienną „book_found” o wartości „False” i przechodzimy do iteracji, najpierw po wartościach listy, a następnie po kluczach i wartościach słownika. Jeśli podana przez użytkownika książka znajduje się w danym słowniku, a jej status to pożyczona („Lent”), to usuwamy status i wyświetla się komunikat, że książka została zwrócona pomyślnie, a wartość zmiennej „book_found” zmienia się na „True”. Następnie zwracamy listę książek i użytkownik widzi pomyślny komunikat, jeśli zmienna „book_found” nadal ma status „False”, użytkownik dostaje komunikat o nieznalezieniu książki. To samo działanie wykonujemy na liście z książkami załadowanymi z pliku.

- „Lend books”, po wybraniu tej opcji zostaje wykonana metoda „lend_book(self)”. W tej metodzie pytamy użytkownika o tytuł książki pożyczonej komuś, imionia osoby, której książkę pożyczyliśmy oraz daty, kiedy została pożyczona. Tu również posługujemy się zmienną typu boolean, a następnie iterujemy najpierw po wartościach listy i przechodzimy do iteracji po słownikach. Jeśli w danym słowniku występuje wybrana książka, ustalamy jej status na „Lent”, przypisujemy imię osoby i datę do odpowiednich kluczy i wyświetlamy informację o pomyślnym zakończeniu akcji. Jeśli użytkownik nie poda imienia lub daty, zachowujemy pustego stringa. Zwracamy listę „self.books” lub „self.loaded_books”. Jeśli zmienna boolean ma status „False” użytkownik widzi informację o nieznalezieniu książki w liście.

- „Read books”, ta opcja wykonuje metodę „read_books(self)”, w której użytkownik najpierw musi podać tytuł przeczytanej książki, następuje automatyczna zamiana pierwszej litery na wielką, następnie ponownie inicjalizujemy zmienną typu boolean –

„False” i iterujemy po liście przechodząc do iteracji po słowniku. Jeśli w danym słowniku znajduje się tytuł, status ustalany jest na „Read”, użytkownik otrzymuje pomyślny komunikat i zmienna „book_found” zmienia wartość na „True”. Ten sam algorytm działania stosujemy w przypadku listy książek dodanych z pliku z poprzedniej sesji użytkownika. Jeśli natomiast zmienna typu boolean ma dalej wartość „False” użytkownik widzi informację o nieznalezieniu książki.

Za pomocą opcji „Save to file” znajdującej się w konstruktorze klasy „BookShelf” użytkownik ma możliwość zapisu zmian dokonanych w załadowanych książkach. Operacja wykonuje się przy pomocy metody „save_to_file(self)”, w której użytkownik podaje ścieżkę do zapisu i następnie otwierany jest plik o podanej ścieżce w trybie zapisu i za pomocą metody „dump()” zostaje zapisana zawartość obiektu „self.books” do tego pliku w formacie „.json”, szerokość znaków ustaliłyśmy na 2 dla większej czytelności pliku. W przypadku wystąpienia błędu, użytkownik otrzyma informację. Jeśli operacja przebiegnie pomyślnie, metoda zwraca „self.loaded_books”, jeśli wystąpi błąd zostanie zwrócone „None”.

W konstruktorze znajduje się również opcja „Load from file”, dzięki której wykonujemy metodę „load_result(self)”. Na początku użytkownik jest proszony o podanie ścieżki do pliku, następnie plik jest otwierany w opcji czytania i dane z pliku są wczytywane przy użyciu funkcji „json.load()” i przypisywane do atrybutu „self.loaded_books” w instancji klasy. Użytkownik dostaje informację o pomyślnym wczytaniu danych i następnie zostają one zwrócone. Pozostałe klauzule „except” obsługują różne rodzaje możliwych do wystąpienia błędów, takich jak brak pliku, błąd dekodowania JSON lub ogólny błąd podczas wczytywania danych, przy wystąpieniu któregoś z błędów metoda zwraca „None”.

`if name == ' main '

Warunek ten tworzy instancję klasy `App` i uruchamia główną pętlę aplikacji.

Cześć III. Działanie kodu na przykładzie (bez wizualizacji)

1. Uruchamiamy aplikację, która wyświetli menu główne z opcjami:

   [I] Info

   [M] More

   [Q] Quit

2. Wybieramy opcję "I" (Info), co spowoduje wyświetlenie informacji o aplikacji:

   This is app Wrap up the chaos!, version 2024.IV.

   App was created to help you manage your books.

3. Następnie, wybieramy opcję "M" (More), co spowoduje wyświetlenie nowych opcji:

   [B] Books

[MV] Movies

4. Wybieramy opcję "B" (Books), co spowoduje uruchomienie klasy związanej z zarządzaniem książkami.

5. W obszarze zarządzania książkami, możemy wybierać różne operacje, takie jak dodawanie nowych książek, usuwanie książek, wyświetlanie kolekcji, kategoryzacja (np. wypożyczenie, zwrócenie, przeczytanie) oraz zapisywanie i wczytywanie danych z pliku.

Część IV. Opis i algorytm działania programu

Graficzną wizualizacje naszej aplikacji zdecydowałyśmy się tworzyć z modułu “CustomTkinter” oraz kilku opcji dostępnych w module “Tkinter”. Moduł “CustomTkinter” wczytałyśmy jako “ctk” w celu większej przejrzystości kodu. Chciałyśmy, aby nasza aplikacja była prosta w obsłudze i optymalna dla każdego użytkownika. Jej charakterystycznymi cechami jest przejrzystość oraz nowoczesny wygląd.

Klasa Gui:

Jest to najważniejsza klasa w aplikacji, która pozwala użytkownikowi wybrać to czym będzie chciał zarządzać. Jest ona bardzo podobna do klasy „App()” z poprzedniej części. Najpierw zaczynamy od stworzenia okna, w tym celu w konstruktorze klasy tworzymy instancję obiektu o nazwie „root” przy użyciu klasy „ctk.CTk()”. „Root” można rozumieć jako wnętrze naszej klasy, czyli nasze okno. Następnie ustalamy jego nazwę oraz wielkość. W konstruktorze znajdują się również dwa zapisy: „self.toplevel_window = None” oraz „self.movies_window = None” niezbędne w dalszej części kodu do otwierania nowych okien.
W dalszej części tworzymy etykietę z tekstem „What do you want to wrap up?", używając klasy „CTkLabel”, etykietę przypisujemy do atrybutu label w obiekcie „self.root” - określamy jej czcionkę (font), kolor tła (fg_color), a ostatnim etapem jest używcie metody „pack()”, aby umieścić ją w oknie, dodatkowo używamy metody „pady=10”, aby umieścić etykietę w odległości 10 pikseli od górnej części okna.

Następnie tworzymy 4 przyciski („Books", „Movies", „Info", „Quit”). Wszystkie przyciski, które znajdują się w aplikacji stworzone są za pomocą tego samego schematu. Budowę omówię na przykładzie jednego przycisku:
„self.buttonB = ctk.CTkButton(self.root, text='Books', font=('Arial', 16), command=self.open_books, fg_color="DeepPink")
self.buttonB.pack(pady=5)”
Najpierw tworzymy przycisk o nazwie „buttonB” za pomocą klasy „CTkButton”. Umieszczamy go w „self.root”, czyli naszym oknie klasy “Gui()”. Określamy tekst, który ma się na nim wyświetlać oraz rodzaj, wielkość czcionki (aby aplikacji była przejrzysta zdecydowałyśmy się na ujednolicenie czcionki i wielkości) oraz kolor tła (również dla ujednolicenia zdecydowałyśmy się na wybranie jednego koloru). Dzięki atrybutowi „command” określamy funkcję, którą ma pełnić dany przycisk, w naszym przypadku wywołuje funkcję „open_book”. Aby przycisk był widoczny w oknie, używamy „pack()” i również dodajemy “pady=5” w celu ułożenia go w odpowiednim miejscu. Analogicznie tworzymy pozostałe przyciski, różnią się one funkcjami, które będą wykonywane po ich naciśnięciu.

Ostatnią metodą, którą wykorzystujemy w konstruktorze klasy „Gui()” jest „mainloop()”. Uruchamia ona główną pętlę programu, co pozwala na interakcję z interfejsem graficznym. Program będzie działał, dopóki użytkownik nie zamknie okna.

Metodę „information(self)” wywołujemy po naciśnięciu przycisku “Info”. Wyświetla ona informacje dotyczące aplikacji (info_message). Wykorzystuje do tego metodę „messagebox.showinfo()” z modułu „tkinter” do pokazania okna informacyjnego z zadanym tekstem.

Metoda „open_books(self)” odpowiada za otwieranie okna związanego z obsługą książek w interfejsie graficznym. Sprawdza warunek, czy zmienna „self.toplevel_window” jest równa „None” (czyli, czy okno jeszcze nie zostało utworzone) lub czy okno już istnieje, ale zostało zamknięte (sprawdzane za pomocą winfo_exists()). Na tej samej zasadzie działa metoda „open_movies(self)”, tylko otwiera okno Movies. Przypisanie wartości „None” dla „self.toplevel_window” oraz „self.movies_window” zostało wykonane w konstruktorze. Jeśli któryś z powyższych warunków jest spełniony, to tworzymy instancję klasy „self.toplevel_window = Book(self.root)” - tworzymy nowe okno, korzystając z klasy „Book” i przypisujemy je do atrybutu „toplevel_window”. Jest to okno „toplevel", które może być najwyższym poziomem w hierarchii okien. Klasy „Book” oraz „Movies” mają swoje nazwy na przestrzeń znajdującą się w nowym oknie, odpowiednio: master oraz Child. Po utworzeniu instancji danej klasy wszystkie metody wykonywane są na tych przestrzeniach. Następnie wykonujemy na nich metodę „mainloop()”, aby uruchomić główną pętlę programu. Dodatkowo stosujemy metodę „grab_set()”, co oznacza, że dane okno ma przechwycić wszelkie zdarzenia wejściowe, czyli użytkownik nie będzie w stanie podjąć interakcji z innymi oknami, dopóki to nie zostanie zamknięte. Ostatnią metodą, którą stosujemy jest „focus()”, który sprawia, że okno będzie aktywne i przyjmuje interakcję od użytkownika. Jeśli początkowy warunek nie zostaje spełniony, czyli okno już istnieje i nie zostało zamknięte, to ustawia fokus na istniejącym oknie.

Metoda „quit(self)” zostaje wywołana po naciśnięciu przycisku „quit”. Zamyka główne okno aplikacji (self.root) poprzez wywołanie metody „destroy()”. To kończy działanie programu i zamyka aplikację.

Klasa Book:

Konstruktor klasy „Book”, przyjmuje argument „root”, który jest referencją do głównego okna interfejsu graficznego, w którym zostanie utworzone nowe okno związanego z obsługą książek. Następnie tworzymy nowe okno toplevel (CTkToplevel) i przypisujemy je do atrybutu master (self.master = ctk.CTkToplevel(root)). W konstruktorze ustawiamy nazwę okna oraz jego wielkość używając atrybutów tiule oraz geometry. Inicjalizujemy również atrybut „categorize_window” na wartość „None”, który będzie niezbędny do powstania następnego okna. W tym konstruktorze również tworzymy etykietę oraz przyciski w taki sam sposób jaki został już przedstawiony. Dodatkowo inicjalizujemy atrybut „books” oraz „loaded_books” jako puste listy.

W tej klasie tworzymy 7 przycisków: Add new books, Remove Book, Show Books, Load from file, Categorize, Save to file, Go back. Budowa przycisków została przedstawiona wcześniej.

Metoda „onbutton(self)" zostaje wywołana po naciśnięciu przycisku „Go back". Używamy w niej metody „destroy()", która zamyka okno master, związane z obsługą książek.

Metoda „open_categorize(self)" otwiera okno do kategoryzacji (CategorizeWindow), w ten sam sposób, w który było otwierane okno „Book": jeśli okno jeszcze nie istnieje lub zostało zamknięte, tworzy nowe okno, ustawia na najwyższym poziomie i aktywuje focus. W przeciwnym razie, jeśli okno już istnieje, ustawia focus na nim.

Pozostałe metody nie zostały zmienione pod względem funkcjonalności w porównaniu z częścią bez wizualizacji. Jedyną różnicą jest brak interakcji z użytkowaniem poprzez konsolę. Wszystkie „input'y" zostały zamienione na „simpledialog.askstring", a „print'y" na „messagebox.showinfo", dzięki temu interakcja z aplikacją dla użytkownika jest bardziej intuicyjna. Jedynie informacje o błędach zostawiłyśmy do wyświetlania się na konsoli.

Klasa CategorizeWindow:

Związana jest z kategoryzacją książek. Konstruktor klasy przyjmuje trzy argumenty: „master" - referencje do okna interfejsu graficznego klasy Book, „books" - listę książek, która będzie przekazana z okna Book, „loaded_books" - lista wczytanych książek, również przekazana z okna Book. Następnie tworzymy nowe okno toplevel (CTkToplevel) i przypisujemy je do atrybutu „master2". W tej klasie również ustawiamy tytuł okna oraz jego wielkość. W następnej części konstruktora przypisujemy listę książek do atrybutu books w obiekcie tej klasy (self.books = books), tak samo robimy z książkami wczytanymi. W tym oknie również tworzymy etykietę oraz przyciski: Lend Books, Return Book, Read Book, Read Book, Borrow Book oraz Go back. Wszystkie te przyciski wywołują metody im odpowiednie, które zostały opisane we wcześniejszej części. Tutaj również zastosowaliśmy „simpledialog.askstring" oraz „messagebox.showinfo".

Klasa Movies:

Konstruktor klasy „Movies", przyjmuje jeden argument „root", który jest referencją do głównego okna interfejsu graficznego. Tworzymy nowe okno toplevel (CTkToplevel) i przypisujemy je do atrybutu „child". Jak w poprzednich klasach ustawiamy odpowiedni tytuł oraz dopasowujemy wielkość okna. Inicjalizujemy atrybut „categorize" na wartość „None". W konstruktorze inicjalizujemy również atrybuty „movies" i „loaded_movies" jako puste listy.

W tej klasie tworzymy 6 przycisków: Add movies, Show movies, Load from file, Categorize, Save to file, Go back. A ich budowa została wcześniej przedstawiona. Naciśnięcie przycisku wywołuje wykonanie metody odpowiadającej nazwie. Wszystkie metody zostały omówione we wcześniejszej części sprawozdania. Metoda „open_categorize" otwiera okno do kategoryzacji (Categorize), w taki sam sposób jak zostały otwierane poprzednie okna. Jeśli

okno jeszcze nie istnieje lub zostało zamknięte, tworzy nowe okno, ustawia na najwyższym poziomie i aktywuje focus. W przeciwnym razie, jeśli okno już istnieje, ustawia focus na nim.

Klasa Categorize:

Jest to klasa do kategoryzacji filmów, która dziedziczy po CTkToplevel z modułu „CustomTkinter". Konstruktor klasy „Categorize" przyjmuje trzy argumenty: „child" - referencja do okna nadrzędnego, które jest obiektem klasy Movies, „movies" - liste filmów, która będzie przekazana z okna Movies oraz „loaded_movies" - listę wczytanych filmów z klasy Movies. Zapis „super().__init__(child)" wywołuje konstruktor klasy nadrzędnej, inicjalizując nowe okno toplevel (CTkToplevel) z oknem nadrzędnym child. Ustawiamy tytuł okna jako "Categorize Movies" oraz jego wielkość. W oknie znajduje się etykieta oraz 2 przyciski: Watched oraz Go back. Przycisk Watched wywołuje metodę "watch_movie", która została opisana w poprzedniej części sprawozdania. Przycisk Go back, który wywołuje metodę "goback", która zamyka okno używając "self.destroy()".

Warunek „ __name__ == "__main__"" został omówiony wcześniej. We wnętrzu znajduje się ustawienie globalnie trybu ciemnego dla aplikacji poprzez użycie: „ctk.set_appearance_mode("dark")". Tutaj tworzymy instancję klasy Gui(), co oznacza, że inicjalizujemy obiekt tej klasy. To jest rozpoczęcie działania aplikacji GUI.

Część V. Podsumowanie

Po skończeniu projektu mamy pewne przemyślenia związane z programowaniem obiektowy z GUI. Od strony programistek uważamy, że programowanie obiektowe, jest złożonym sposobem programowania, które ma swoje zasady, ale po ich zrozumieniu, jest bardzo przydatne w tworzeniu programów. Jednak osobiście uważamy, że wolimy wersję aplikacji bez GUI, ponieważ nie musimy zastanawiać się nad kolorami, tłem czy rozmiarem czcionki lub okna, żeby wszystko pasowało do siebie, a okna się nie nakładały. Rozumiemy jednak aspekt przeciętnego użytkownika, który woli, aby była graficzna wizualizacja, ponieważ widzi tylko efekt końcowy, który rozumie i odpowiada jego oczekiwaniom. Uważamy, że wiedza z GUI może być przydatna w momencie, w którym pracujemy w firmie lub przy projekcie, którego interfejs będzie użytkowany przez klientów, jeśli jednak zajmujemy się pracą z danych czy ich analizą, to wolimy wersje aplikacji bez graficznej wizualizacji.