Gabriela Zapała gl. 07

Projekt z przedmiotu Języki Symboliczne

Dokumentacja

1. Temat projektu - Automat biletowy MPK

Automat przechowuje informacje o monetach/banknotach znajdujących się w nim (1, 2, 5, 10, 20, 50gr, 1, 2, 5, 10, 20, 50 zł) [dziedziczenie: można napisać uniwersalną klasę PrzechowywaczMonet po której dziedziczyć będzie automat]. Okno z listą biletów w różnych cenach (jako przyciski).
Wymagane bilety: 20-minutowy, 40-minutowy, 60-minutowy w wariantach normalnym i ulgowym.
Możliwość wybrania więcej niż jednego rodzaju biletu. Możliwość wprowadzenia liczby biletów. Po wybraniu biletu pojawia się okno z listą monet (przyciski) oraz możliwością dodania kolejnego biletu lub liczby biletów. interfejs ma dodatkowo zawierać pole na wybór liczby wrzucanych monet (domyślnie jedna). Po wrzuceniu monet, których wartość jest większa lub równa cenie wybranych biletów, automat sprawdza czy mogę wydać resztę:
Brak reszty/może wydać: wyskakuje okienko z informacją o zakupach, wydaje resztę (dolicza wrzucone monety, odlicza wydane jako reszta), wraca do wyboru biletów.
Nie może wydać: wyskakuje okienko z napisem "Tylko odliczona kwota" oraz zwraca wrzucone monety.

2. Implementacja - https://github.com/gzapalka/projektPython

Klasa Coin:

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/coin.py#L10

Reprezentuje pieniądze przechowywane i wrzucane do automatu biletowego. Każdy obiekt typu Coin posiada atrybut value nadawany mu w konstruktorze, symbolizujący wartość pieniądza.

Zawiera funkcje:
`__init__` tworzącą nowy obiekt typu Coin i nadający mu wartość podaną w parametrze
`get_value` zwracający wartość monety
`__lt__` i `__eq__` umożliwiającą porównywanie dwóch obiektów typu Coin
`__str__` i `__repr__` określającą w jaki sposób obiekt tworzyć string z obiektów typu Coin

Klasa Ticket:

https://github.com/gzapalka/projektPython/blob/924972336be9a8ddfe6ef4d90cc280971e89e459/ticket.py#L9

Reprezentuje bilety możliwe do zakupienia w automacie. Każdy obiekt typu Ticket posiada atrybut price (symbolizujący jego cenę) i description (zawierający opis - rodzaj, ilość minut podróży do których upoważnia).

Zawiera funkcje:
`__init__` tworzącą nowy obiekt typu Ticket i nadający mu cenę i opis podane jako parametry
`get_price` zwracająca cenę biletu
`get_ticket_description` zawierająca opis biletu
`__str__` i `__repr__` określającą w jaki sposób obiekt tworzyć string z obiektów typu Ticket

Klasa Payment_Management:

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/payment_managment.py#L11

Zajmuje się logiką zarządzania pieniędzmi. Zawiera metody operujące na klasie Coin i inne związane z pieniędzmi.

Zawiera metody:

`__init__(self)` Tworzy obiekt typu Payment Management i i listy `expected_value` zawierającą wartości `(Decimal) Coin` na jakich operuje i lista zawierającą wszystkie monety które w danym momencie posiada.

`add_coin(self, value: float)` Tworzy obiekt typu `Coin` i dodaje go do listy przechowywanych.

`get_sum(self)` Zwraca sumę wartości obecnie posiadanych monet.

`return_coin(self, coin: Coin)` Usuwa monetę równą podanej z listy jeżeli się na niej znajduje. W przeciwnym wypadku zwraca `False.`

`change(self, amount: Decimal)` Wydaje resztę. Zwraca listę monet stanowiących resztę i usuwa je z posiadanych. Jeżeli wydanie reszty nie jest możliwe zwraca `False.`

`get_payment(self)` Zwraca łączną wartość wrzuconych monet

`get_credit(self)` Zwraca wartość należnej zapłaty

`get_amount_to_pay(self)` Zwraca wartość należnej reszty

`clearList(self)` Usuwa wszystkie przechowywane monety

Klasa Handler:

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/handler.py#L19

Zajmuję się logiką działania automatu biletowego. Dziedziczy z klasy Payment_Management do obsługi operacji na pieniądzach i obsługuje pozostałe operacje na biletach.

Zawiera metody:

`__init__(self)` Tworzy obiekt typu handler i atrybuty: listę ticket_values zawierającą ceny dostępnych biletów, listę bought tickets zawierającą bilety, które mają zostać zakupione w obecnej transakcji i added_coins zawierające monety wrzucone przez klienta.

`create_add_ticket(self, value: float, description: str)` Zwraca funkcję naliczającą opłatę za konkretny bilet

`check_data(self, entry: list)` Sprawdza czy podana została poprawna (naturalna) liczba monet .

`resign(self)` Resetuje automat, usuwa dane obecnej transakcji.

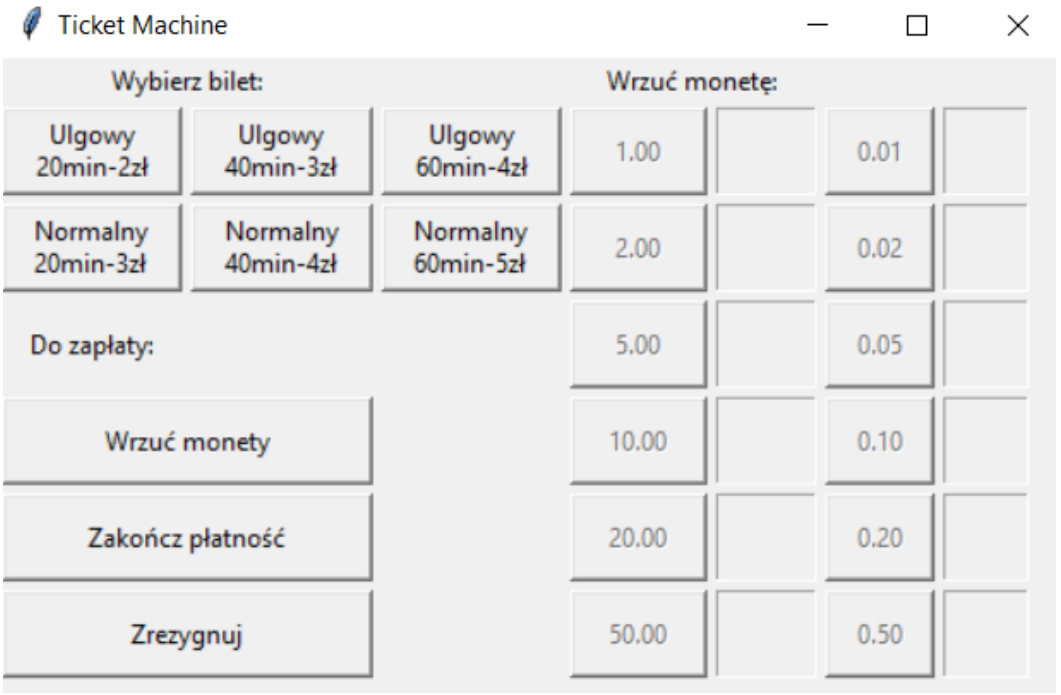
`insert_coins(self, entry: list, gui = None)` Dodaje wrzucone monety do reszty przechowywanych przez automat.

`manage_change(self, entry: list, gui = None)` Zwraca resztę jeżeli jest potrzebna lub rzuca wyjątek jeżeli to niemożliwe. Zwraca listę monet stanowiących resztę obliczoną przez `Payment_Management.change()`, pustą listę gdy reszta nie jest potrzeba, krotkę jeżeli nie jest w stanie wydać reszty. Wyrzuca wyjątek `exceptions.NotEnoughMoney` jeżeli wartość biletów przewyższa wartość wrzuconych monet.

Klasa App:

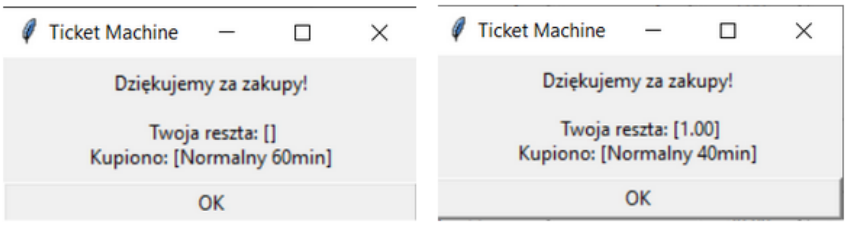
https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/app.py#L15

Tworzy GUI i operuje nad nim wykorzystując klasę Handler. Interfejs użytkownika zawiera:



GUI

Przyciski do wyboru biletów obsługujące funkcję add_ticket utworzone przez funkcję Handler.create_add_ticket
Widgety typu entry do wpisanie liczby monet danej wartości jaką chcemy wrzucić. Pisanie do nich jest możliwe dopiero po wybraniu przynajmniej jednego biletu.
Przycisk "Wrzuć monety" dołączające podane przez użytkownika monety do zbioru.
Przycisk "Zakończ płatność" dołączające podane przez użytkownika monety do zbioru i wydający resztę, podający informację o przebiegu transakcji.
Przycisk "Zrezygnuj", który resetuje automat usuwając informacje o obecnej transakcji.



Przykładowe wyniki poprawnej transakcji.

Aby realizować swoje zadanie klasa App zawiera metody:

`__init__` Tworzy obiekt typu app i okno aplikacji.
`display_screen` Tworzy i ustawia widgety wyświetlane na `GUI.`
`update_info` Uaktualnia informacje o łącznej cenie biletów.
`clear_entries` Usuwa dane z widgetów typu `entry`
`create_entries` Tworzy widgety typu entry
`place_entries` Układa na oknie widgety typu `entry`
`create_keypad` Układa na oknie przyciski
`switch_entries_enabale(self,display: bool)`
Włącza/wyłącza możliwość wpisywania do widgetów typu entry.
`read_data_from_entries` Czyta dane z widgetów typu entry
`insert_coins` Obsługuje przycisk **"`Wrzuć monety`"**.
`resign(self)` Obsługuje przycisk **"`Zrezygnuj`"**. Resetuje GUI, usuwa dane o obecnej
`confirm_payment` Obsługuje przycisk **"`Zakończ płatność`"**. Uruchamia procedurę kończenia transakcji i wyświetla informacje o jej przebiegu.

`popup_window(self, message: str)`

Tworzy okna typu pop-up dla wyników `confirm_payment`.

3. Testy:

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/test_handler.py#L15

Test 1. Bilet kupiony za odliczoną kwotę - oczekiwany brak reszty.

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/test_handler.py#L26

Test ustawia wartość długu na 5 zł i sprawdza czy dla wrzuconej monety pięciozłotowej automat nie z wraca reszty.

Test 2. Bilet kupiony płacąc więcej - oczekiwana reszta.

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/test_handler.py#L19

Test ustawia wartość długu na 5 i sprawdza czy dla wrzuconych monet 5 zł i 2 zł reszta wyniesie 2 zł.

Test 3. Bilet kupiony płacąc więcej, automat nie ma jak wydać reszty - oczekiwana informacja o błędzie oraz zwrócenie takiej samej liczby monet o tych samych nominałach, co wrzucone..

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/test_handler.py#L34

Test ustawia wartość długu na sześć i sprawdza czy dla wrzuconych monet 5 zł i 2 zł Handler zwraca krotkę z pierwszym elementem False i drugim elementem będącym listą monet o tych samych nominałach co wrzucone.

Test 4. Zakup biletu płacąc po 1 gr - suma stu monet 1gr ma być równa 1 zł.

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/test_handler.py#L43

Test ustawia wartość długu na 1 zł, wrzuca 100 razy jeden grosz i sprawdza czy automat nie zwraca reszty.

Test 5. Zakup dwóch różnych biletów naraz - cena powinna być suma

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/test_handler.py#L54

Test dodaje do rachunku dwa bilety: jeden o cenie 5 i jeden o cenie 3 po czym sprawdza czy ich łączna cena jest poprawnie obliczona.

Test 6. Dodanie biletu, wrzucenie kilku monet, dodanie drugiego biletu, wrzucenie pozostałych monet, zakup za odliczoną kwotę - oczekiwany brak reszty (wrzucone monety nie zerują się po dodaniu biletu).

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/test_handler.py#L61

Test dodaje do rachunku bilet o wartości 3 zł, wrzuca monetę o wartości 1, dodaje bilet o wartości 3 i wrzuca monetę o wartości 5 po czym sprawdza czy automat nie zwraca reszty.

Test 7. Próba wrzucenia ujemnej oraz niecałkowitej liczby monet (oczekiwany komunikat o błędzie).

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/test_handler.py#L71

Test sprawdza czy dodanie niecałkowitej lub ujemnej liczby monet wyrzuca wyjątek `exceptions.InvalidArgument`.

Wszystkie testy kończą się pomyślnie.

```
(base) C:\Users\Gabriela\Desktop\studia\sem4\JS\projekt_zaliczeniowy>python test_handler.py
Coin imported succenfully
Management Payment imported succesfully
App imported succesfully
.......
----------------------------------------------------------------------
Ran 7 tests in 0.012s

OK
```

4. Istotne fragmenty kodu:

Lambda-wyrażenia

https://github.com/gzapalka/projektPython/blob/924972336be9a8ddfe6ef4d90cc280971e89e459/app.py#L157

Lambda użyta w tym fragmencie umożliwia przypisanie dwóch metod do przycisku, co gwarantuje zmianę informacji o należnej opłacie od razu po jej zmianie.

List comprehensions

https://github.com/gzapalka/projektPython/blob/924972336be9a8ddfe6ef4d90cc280971e89e459/app.py#L132
Wartości wpisane do widgetów typu entry pakowane są do listy. Jeżeli przeczytany jest pusty znak zastępujemy go ciągiem 0.

https://github.com/gzapalka/projektPython/blob/924972336be9a8ddfe6ef4d90cc280971e89e459/app.py#L138
Tworzy i pakuje do listy potrzebną ilość widgetów typu entry.

https://github.com/gzapalka/projektPython/blob/924972336be9a8ddfe6ef4d90cc280971e89e459/app.py#L154
Tworzy listę złożoną z funkcji utworzonych przez `create_add_ticket` z parametrów zapisanych w `ticket_values` i `available_tickets`.

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/handler.py#L28
Tworzy listę wartości typy Decimal obsługiwanych biletów.

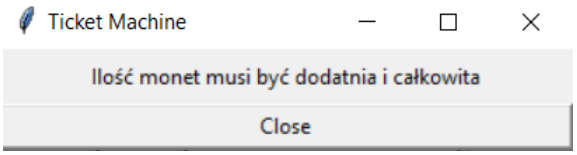
Klasy

Zostały wyszczególnione w 2 punkcie.

Wyjątki: Plik exceptions:

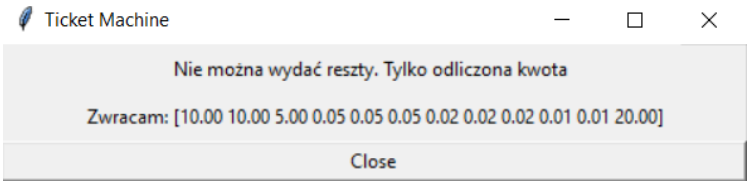
https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/exceptions.py#L13
Zawiera wyjątki utworzone na potrzeby projektu:

`InvalidArgument` Informujący o próbie wrzucenia niepoprawnej liczby monet (ujemnej lub niecałkowitej)



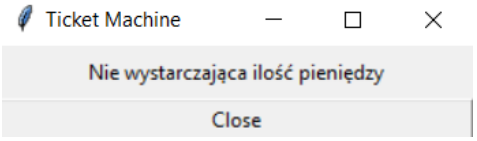
pop-up dla InvalidArgument

`CannotChange` Informujący, że automat nie jest w stanie wydać wymaganej reszty.



pop-up dla CannotChange

`NotEnoughMoney` Informujący, że koszt biletów przewyższa wartość wrzuconych monet.



pop-up dla NotEnoughMoney

Zawiera również metodę `popup_window` używaną przez wyjątki to tworzenia okna typu pop-up z informacją o zaistniałym błędzie.

Moduły:

Program podzielony jest na trzy główne moduły.

`App` - zajmujący się obsługą interfejsu
https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/app.py#L15

`Handler` - obsługujący logikę działania automatu
https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/handler.py#L19

`Payment_Management` - Zajmuje się logiką zarządzania pieniędzmi.
https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/payment_managment.py#L11

Dekoratory:

https://github.com/gzapalka/projektPython/blob/12152838c87e9573caf7188adcd97f35151dcd24/handler.py#L32

Metoda zwraca nowo utworzoną funkcję `add_ticket` dla podanej wartości.