Tomasz Bortacki GL.01

Projekt z Języków symbolicznych

Dokumentacja

Temat projektu:

Tematem projekty jest **Automat z napojami**. Program symuluje pracę działania automatu, który wydaje napoje. Jesteśmy w stanie wybrać produkt w zakresie od 30 do 50, a następnie za pomocą przycisków wrzucić odpowiednie monety od 1gr do 5zł. Program wydaje produkt oraz wydaje resztę w momencie, gdy wrzuci się za dużo. Zwraca też same wrzucone pieniądze w momencie, gdy przerwiemy transakcję

Funkcjonalność

1. Program posiada duży ekran, na którym wyświetla się w zależności od stanu automatu:
    - Wyświetla informacje o wybraniu produktu
    - Ilość wrzuconych monet, gdy produkt zostanie wybrany oraz cene produktu
2. Pod ekranem znajdują się przyciski **0-9** oraz **CLS** i **OK**
3. **CLS** jest używane do przerywania transakcji oraz czyszczenia ekranu, gdy wybierzemy zły produkt
4. **OK** zatwierdza wybranie produktu i przełącza automat w tryb wrzucania monet
5. W momencie, gdy automat jest w stanie wrzucania monet, to przyciski **0-9** oraz **OK** zostają wyłączone i włączają się przyciski, które są z prawej strony od ekranu i odpowiadają monetom od 1gr do 5zł
6. Po wrzuceniu odpowiedniej lub za dużej ilości monet zostaje wydany produkt oraz zwrócona reszta, gdy takowa się należy. Pojawia się odpowiedni komunikat z informacją o tym
7. Jeżeli stwierdzimy, że chcemy przerwać transakcje, to po wciśnięciu przycisku **CLS** zostaje nam zwrócona ilość monet, która została wrzucon do automatu. Pojawia się odpowiedni komunikat

Klasy i samodzielne funkcje zawarte w projekcie

Klasa main

Klasa odpowiada za włączenie automatu, wyświetla odpowiedni komunikat w konsoli oraz załącza klasę **gui**

Klasa gui

Klasa odpowiada za wyświetlenie interfejsu automatu oraz inicjuje wszystkie interakcje w automacie. Możemy w niej wybierać produkty oraz wrzucać odpowiednie monety. Zostają one obsłużone i przekazane do klasy **core**, w której dostajemy odpowiednie produkty, informacje, o stanie, ich braku, czy też, że produkt został wydany

- init: wykorzystując bibliotekę **tkinter** tworzymy podstawowe wartości dla ekranu oraz z tworzymy obiekt automatu z klasy **core**
- design: Tutaj inicjujemy wszystkie metody, które są odpowiedzialne za utworzenie layoutu automatu oraz tworzy siatkę, w której siedzą odpowiednie guziki i ekran
- add_screen: metoda odpowiedzialna za dodawanie ekranu. Tworzy trzy instacje **Label** z tkintera. Pierwsza zapobiega przesuwaniu się ekrany, druga dodaje ekran główny, na którym wyświetlają się najważniejsze informacje oraz trzecia, która wyświetla cene wybranego produktu
- add_buttons: Tworzy tablice przycisków **-09**, **CLS** oraz **OK** Za pomocą **List Comprehension**, oraz ustawia je w odpowiednich miejscach w siatce ekranu. Każdy z przycisków używa **lambda** To wywoływania metod, które są do nich przypisane z wartościami, które ma każdy przycisk osobne
- add_coins: Robi dokładnie to samo co metoda **add_coins**, jednak w tym przypadku tworzy tablicę przycisków z monetami. Każdy z przycisków używa **lambda** To wywoływania metod, które są do nich przypisane z wartościami, które ma każdy przycisk osobne
- choose_product: Metoda przypisana do przycisków, które tworzy **add_buttons**, odpowiedzialna za wybór produktu oraz przerywanie transakcji
- clear_screen: Metoda odpowiedzialna za czyszczenie ekranu w przypadku, gdy zostanie przerwana transakcja lub w trakcie płacenia wystąpi błąd
- set_price: Metoda odpowiedzialna za wyświetlenie ceny produktu w przypadku, gdy zostanie któryś wybrany
- find_product: Szuka produktu i jeżeli go znajdzie, to przełącza automat w tryb wrzucania monet. Następnie pokazuje ile pieniędzy sumarycznie zostało wrzuconych do automatu. W przypadku błędów wyświetla je nam na ekranie
- pay: Metoda przypisana do guzików, które tworzy metoda **add_coins**. Odpowiada za płacenie danym typem monety i jeżeli zostanie wrzucona odpowiednia ilość lub więcej za produkt, to automat wydaje produkt oraz resztę. W przypadku błędów wyświetla je nam na ekranie

Klasa core

Metoda generuje nam tablice z produktami i ich cenami, które są przechowywane następnie w klasie **assortment**. Mamy w niej metody, które odpowiadają za sprawdzanie stanu, wprowadzanie monet i wydawanie produktów. W przypadku błędów są one zwracane w tej klasie do klasy **gui** i wyświetlane użytkownikowi

- price_generator: Generator, poza klasą główną, służy do generowania tablicy produków o losowej cenie w podanym przedziale
- init: Metoda klasy, odpowiedzialna za generowanie produków, które następnie tworzą w głównym konstruktorze klasę i odpowiednie metody, które są do niej potrzebne
- get_product_price: Metoda wyszukuje produktu o podanym id oraz zwraca jego cenę
- pay: Metoda odpowiedzialna za płacenie. Wrzuca monety do automatu i sprawdza, czy została wrzucona odpowiednia ilość monet. jeżeli tak, to zwraca produkt i resztę
- clear: Metoda czyści transakcje oraz zwraca wrzucone monety, jeżeli została ona wcześniej przerwana
- get_money: Metoda zwraca wartość wrzuconych monet
- get_product_and_rest: Metoda wydaje produkt oraz resztę i wyrzuca błędy, jeżeli jakieś napotka

Klasa assortment

Przechowujemy w niej produktach oraz ich ilościach. Wykorzystujemy do tego klasę **wrapper**. Posiada ona metodę, która zwraca informacje o stanie produktu o podanym ID. Przez to jesteśmy w stanie stwierdzić, czy produkt się nie skończył. Sama klasa korzysta z **List comprehension** do utworzenia tablicy produktowej, która została przekazana z **core**, zaczynając podstawowo od indexu = 30

- get_qty: Metoda zwraca informacje i stanie produktów o podanym id, czy przypadkiem się nie skończył

Klasa bank

Przechowujemy tutaj informacje o stanie monet. Klasa jest wykorzystywana do przechowywania monet w automacie, jak i tych, które zostają wrzucone w trakcie wydawania produktu. Wykonują się tutaj akcje jak czyszczenie banku, łączenie dwóch ze sobą (Dodaje monety do siebie, jeżeli transakcja dojdzie do skutku). Tak jak w przypadku klasy **assortment* wykorzystuje ona klasę **wrapper**

- **add**: Metoda, która dodaje dwa obiekty do siebie. Dodając tym samym nowe monety, po zatwierdzeniu opłaty za produkt
- change: Metoda klasowa odpowiedzialna za przypisanie ilości danej waluty, jeżeli nie zostanie podany parametr to metoda tworzy pusty bank
- set: Metoda dodaje monety do odpowiednich miejsc
- load: Metoda po odpowiedniej wartości monety ustawia jej ilość
- get_amount: Metoda zwraca sumę wszystkich monet w banku
- get_rest: Metoda zwraca wrzucone monety.
- get_diff: Metoda w przypadku zatwierdzenia transakcji zwraca resztę

Klasa exceptions

Posiada ona niestandardowe klasy błędów, przez które możemy następnie zwracać informacje o błędach w aplikacji

Klasa items

Ma informacje o pojedynczych rzeczach. Możemy tutaj sprawdzić cenę, czy ilość.

- set_qty: Metoda dodaje odpowiednia ilość monet
- get_qty: Metoda zwraca ilość rzeczy
- get_float_val: Metoda zwraca wartość danej rzeczy w postaci zmiennoprzecinkowej
- get_sum_int_val: Metoda zwraca sumę wszystkich rzeczy w postaci stałej
- get_sum_float_val: Metoda zwraca sumę wszystkie rzeczy w postaci zmiennoprzecinkowej

Klasa money

Ma informacje o danej monecie, takie jak ilość oraz wartość. Możemy w niej również dodawać nowe monety

- set: Metoda ustawia ilość monet
- get: Metoda zwraca ilość monet

Klasa product

Wykorzystuje klasę **items**. Pozwala nam te samą metodę abstrakcyjną oraz metodę, która pozwala nam zwrócić nazwę produktu

- get: Metoda zwraca podaną ilość produktu
- get_name: Metoda zwraca nazwę produktu

Klasa wrapper

Klasa przechowująca słownik przedmiotów z identyfikatorami, dzięki czemu mamy posegregowane przedmioty według ich rodzaju. Pozwala nam usuwać produkty, pobierać nazwę produktu, danego typu, zwracać słownik z przedmiotami oraz pobierać cenę produktów o danym typie

- remove: Metoda usuwa dany przedmiot. Usuwa po wydaniu produktu
- get_name: Metoda zwraca nazwę produktu
- get_info: Zwraca informacje, które są przechowywane w tym obiekcie. Na przykład: Produkty o podanym typie
- get_price: Metoda zwraca cenę przedmiotu o podanym id
- set: Metoda abstrakcyjna zakłada dodanie danej ilości przedmiotów do danego typu danych

Plik ze statycznymi metodami utils

Przechowujemy tutaj metody, które są re używalne, jak na przykład:

- set_proper_text: Ustawia nam odpowiedni tekst na przyciskach
- set_proper_coin: Ustawia odpowiedni typ monety na ekranie. W zależności od wrzuconych monet dobiera odpowiedni format, żeby na ekranie wyświetlało się na przykład: 10gr lub 1.05zł
- change_buttons_state: Metoda odpowiedzialna za włączanie i wyłączanie przycisków
- show_info: Metoda wyświetla modal z informacjami np: Jaki produkt zwrócono, czy resztę
- show_error: Metoda wyświetla modal z błędami np: Złe wykorzystanie metody, wprowadzenie do niej złych parametrów
- multipy Metoda za pomocą lambdy zwraca wartość potęgi

Są tam również metody, które zmieniają stan przycisków w aplikacji oraz metody odpowiedzialne za wyświetlanie błędów czy informacji na ekranie

Plik ze słownikiem dictionary

Ma informacje, które są re używalne jak, tekst, który zmienia się na ekranie, czy kolory aplikacji

Testy

Wykorzystujemy bibliotekę do testów: **unittest**

test_1

Sprawdzenie ceny jednego towaru - oczekiwana informacja o cenie. Sprawdza metodę **get_product_price** z klasy **core**. Zwraca ona cenę w postaci float i porównuje ją do tego co zostało utworzone przed testami

test_2

Wrzucenie odliczonej kwoty, zakup towaru - oczekiwany brak reszty. Wybiera pierwszy produkt o id 30 i wykorzystując metodę **pay** klasy **core** płaci odpowiednimi monetami za produkt, a następnie zwraca produkt. Cena była wyliczona, przez co spodziewamy się, że nie dostaniemy reszty

test_3

Wrzucenie większej kwoty, zakup towaru - oczekiwana reszta. Jak w przypadku wcześniejszego testu, płacimy odpowiednimi metodami, jednak tym razem płacimy więcej i spodziewamy się informacji o reszcie

test_4

Wykupienie całego asortymentu, próba zakupu po wyczerpaniu towaru - oczekiwana informacja o braku. Kupujemy jeden produkt, a następnie drugi, jednak w asortymencie był jednie jeden produkt, przez co za drugim razem spodziewamy się błędu: **EmptyProductError**

test_5

Sprawdzenie ceny towaru o nieprawidłowym numerze (<30 lub >50) - oczekiwana informacja o błędzie. Próbujemy wybrać produkt z zakresu, w którym nie mamy produktów i oczekujemy błędu: **WrongProductError**

test_6

Wrzucenie kilku monet, przerwanie transakcji - oczekiwany zwrot monet. Wybieramy produkt, a następnie za niego płacimy, jednak niewystarczająco i przerywamy transakcję. Oczekujemy informacji o zwrocie wrzuconych monet.

test_7

Wrzucenie za małej kwoty, wybranie poprawnego numeru towaru, wrzucenie reszty monet do odliczonej kwoty, ponowne wybranie poprawnego numeru towaru - oczekiwany brak reszty.

test_8

Zakup towaru płacąc po 1 gr - suma stu monet ma być równa 1zł (dla floatów suma sto razy 0.01+0.01+...+0.01 nie będzie równa 1.0). Płatności można dokonać za pomocą pętli for w interpreterze. Wybieramy produkt o cenie 9.90, a następnie wpłacimy za niego po 1gr. Dajemy metodę **pay** do pętli for, która wykonuje się 989 razy, a następnie płacimy ostatni raz 1gr i oczekujemy, że zostanie zwrócony produkt