POLITECHNIKA ŚWIĘTOKRZYSKA	
Wydział: **WEAiI**	Przedmiot: Technologie obiektowe
Grupa: **1IZ21A**	Temat: **Object CSV Mapper**
Autor: **Dawid Duda**	

Odwzorowanie pomiędzy językiem obiektowym a formatem CSV polega na konwersji danych z jednego formatu do drugiego. CSV jest formatem pliku tekstowego, który przechowuje dane tabelaryczne w postaci wartości oddzielonych przecinkami. Z drugiej strony, języki obiektowe są zorientowane na obiekt i przechowują dane w postaci obiektów, które składają się z właściwości i metod.

Aby odwzorować dane z języka obiektowego na CSV, musimy przekształcić właściwości obiektów na kolumny CSV, a każdy obiekt na wiersz w pliku CSV. Podobnie, aby odwzorować dane z formatu CSV na język obiektowy, musimy odczytać wartości z pliku CSV i utworzyć obiekty na podstawie tych wartości.

Implementacja Object CSV Mappera może obejmować następujące kroki:

1. Określenie struktury obiektów i ich właściwości, które będą mapowane na kolumny CSV.
2. Utworzenie klasy, która będzie mapować obiekty na CSV i odwrotnie. Klasa ta może zawierać metody takie jak to_csv() i from_csv().
3. Implementacja metody to_csv(), która będzie przyjmować obiekt i zwracać wiersz CSV.
4. Implementacja metody from_csv(), która będzie przyjmować wiersz CSV i tworzyć obiekt na podstawie wartości z kolumn CSV.
5. Przetestowanie klasy na przykładzie danych wejściowych.

Ostatecznie, Object CSV Mapper może być przydatny do łatwej konwersji danych między językiem obiektowym a formatem CSV, co ułatwia przechowywanie i przetwarzanie danych tabelarycznych w różnych aplikacjach.

Klasa ObjectCSVMapper służy do mapowania obiektów Pythona na pliki CSV i na odwrót. Klasa pozwala na łatwe zapisywanie i odczytywanie listy obiektów do i z pliku CSV.

Metody

- __init__(self, fields) - Konstruktor klasy ObjectCSVMapper. Przyjmuje listę nazw pól (kolumn) w pliku CSV.
- to_csv(self, obj) - Metoda, która mapuje pojedynczy obiekt na wiersz w formacie CSV. Przyjmuje obiekt jako argument i zwraca listę wartości pól tego obiektu.
- from_csv(self, row, fields) - Metoda, która mapuje pojedynczy wiersz z pliku CSV na obiekt. Przyjmuje listę wartości wiersza i listę nazw pól, a następnie zwraca obiekt utworzony na podstawie tych wartości.
- to_csv_list(self, lst) - Metoda, która mapuje listę obiektów na ciąg znaków w formacie CSV. Przyjmuje listę obiektów jako argument i zwraca ciąg znaków zawierający wiersze CSV dla każdego obiektu.
- from_csv_list(self, csv_str) - Metoda, która mapuje ciąg znaków w formacie CSV na listę obiektów. Przyjmuje ciąg znaków CSV jako argument i zwraca listę obiektów utworzoną na podstawie wierszy CSV.

- to_csv_file(self, objs, filename) - Ta funkcja przyjmuje listę obiektów **objs** oraz nazwę pliku **filename** i zapisuje obiekty do pliku CSV. Działanie funkcji polega na iteracji po obiektach w liście **objs**. Dla każdego obiektu wywoływana jest metoda **to_csv**, która mapuje obiekt na wiersz w formacie CSV. Następnie wiersz ten zostaje zapisany do pliku przy użyciu modułu **csv**. Pierwszy wiersz pliku zawiera nazwy pól (kolumn) określone w klasie **ObjectCSVMapper**.
- from_csv_file(self, filename) - Ta funkcja przyjmuje nazwę pliku **filename** i odczytuje obiekty z pliku CSV. Działanie funkcji polega na odczytaniu zawartości pliku CSV przy użyciu modułu **csv**. Najpierw odczytywane są nazwy pól (pierwszy wiersz pliku), a następnie dla każdego kolejnego wiersza wywoływana jest metoda **from_csv**, która mapuje wiersz na obiekt. Odczytane obiekty są dodawane do listy i zwracane jako wynik funkcji.

Implementacja Object CSV Mapper w języku Python:

```
import csv

class ObjectCSVMapper:
    def __init__(self, fields):
        self.fields = fields

    def to_csv(self, obj):
        row = []
        for field in self.fields:
            value = getattr(obj, field)
            if isinstance(value, list) or isinstance(value, set):
                value = ';'.join(map(str, value))
            elif isinstance(value, dict):
                value = ','.join([f"{k}:{v}" for k, v in value.items()])
            elif isinstance(value, tuple):
                value = ','.join(str(x) for x in value)
            elif isinstance(value, Address):

                value = "$" + value.a_id + "$"
            else:
                value = str(value)
            row.append(value)
        return row

    @classmethod
    def from_csv(cls, row, fields):
        obj = type('', (), {})()
        for i, field in enumerate(fields):
            value = row[i]
            if ';' in value:
                value = value.split(';')
                if len(value) == 1:
                    value = value[0]
            elif ',' in value:
                items = value.split(',')
                if ':' in items[0]:
                    value = {k: v for k, v in [x.split(':') for x in items]}
                else:
```

```
                value = tuple(items)
            elif value.isdigit():
                value = int(value)
            else:
                try:
                    value = float(value)
                except ValueError:
                    pass
            setattr(obj, field, value)
        return obj


    def to_csv_list(self, lst):
        csv_rows = []
        for obj in lst:
            row = self.to_csv(obj)
            csv_rows.append(','.join(row))
        return '\n'.join(csv_rows)

    def from_csv_list(self, csv_str):
        lst = []
        rows = csv_str.strip().split('\n')
        for row in rows:
            row_lst = row.split(',')
            obj = self.from_csv(row_lst, self.fields)
            lst.append(obj)
        return lst

    def to_csv_file(self, objs, filename):
        with open(filename, 'w', newline='') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow(self.fields)
            for obj in objs:
                row = self.to_csv(obj)
                writer.writerow(row)

    def from_csv_file(self, filename):
        objs = []
        with open(filename, 'r', newline='') as csvfile:
            reader = csv.reader(csvfile)
            fields = next(reader)
            for row in reader:
                obj = self.from_csv(row, fields)
                objs.append(obj)
        return objs

class Person:
    def __init__(self, name, age, hobbies, favorite_books, contact_info, address):
        self.name = name
        self.age = age
        self.hobbies = hobbies
        self.favorite_books = favorite_books
```

```
        self.contact_info = contact_info
        self.address = address

class Address:
    def __init__(self, a_id, street, city):
        self.a_id = a_id
        self.street = street
        self.city = city

add = [
      Address("id_1", "Wall st.", "New York"),
      Address("id_2", "Rodeo Drive", "Los Angeles"),
      Address("id_3", "Main st.", "Dallas"),
      ]
mapper1 = ObjectCSVMapper(["a_id", "street", "city"])

# write the objects to a CSV file
mapper1.to_csv_file(add, "address.csv")

people = [
    Person("John", 25, ["reading", "hiking"], {"fiction": "1984", "non-fiction": "The Art of Thinking
Clearly"}, ("john@example.com", "555-1234"), add[0]),
    Person("Jane", 30, ["swimming", "dancing"], {"fiction": "Pride and Prejudice", "non-fiction":
"Sapiens"}, ("jane@example.com", "555-5678"), add[1]),
    Person("Bob", 40, ["gardening", "cooking"], {"fiction": "The Great Gatsby", "non-fiction":
"Atomic Habits"}, ("bob@example.com", "555-9101"), add[2]),
]
# create an instance of ObjectCSVMapper with the desired fields
mapper = ObjectCSVMapper(["name", "age", "hobbies", "favorite_books", "contact_info",
"address"])

# write the objects to a CSV file
mapper.to_csv_file(people, "people.csv")

# read the CSV file back into a list of objects
people_from_csv = mapper.from_csv_file("people.csv")

addresses_from_csv = mapper1.from_csv_file("address.csv")

# print the list of objects
for person in people_from_csv:
    for address in addresses_from_csv:
        if str( "$" + address.a_id +"$" ) == str(person.address):
            person.address = address
            break

# print the list of objects with addresses included
for person in people_from_csv:
    print(person.name, person.age, person.hobbies, person.favorite_books, person.contact_info,
person.address.street, person.address.city)
```

Instrukcja użycia programu:

1. Zdefiniuj listę obiektów, które chcesz zapisać do pliku CSV. Możesz utworzyć obiekty według własnych potrzeb i upewnić się, że mają odpowiednie atrybuty, które będą mapowane na kolumny w pliku CSV.

2. Zdefiniuj obiekt typu ObjectCSVMapper, podając mu listę pól, które chcesz zapisać do pliku CSV. Na przykład:

```
mapper1 = ObjectCSVMapper(["a_id", "street", "city"])
mapper = ObjectCSVMapper(["name", "age", "hobbies", "favorite_books", "contact_info",
"address"])
```

3. Dodaj obiekty do listy, którą chcesz zapisać. Na przykład:

```
add = [
     Address("id_1", "Wall st.", "New York"),
     Address("id_2", "Rodeo Drive", "Los Angeles"),
     Address("id_3", "Main st.", "Dallas"),
     ]

people = [
   Person("John", 25, ["reading", "hiking"], {"fiction": "1984", "non-fiction": "The Art of Thinking
Clearly"}, ("john@example.com", "555-1234"), add[0]),
   Person("Jane", 30, ["swimming", "dancing"], {"fiction": "Pride and Prejudice", "non-fiction":
"Sapiens"}, ("jane@example.com", "555-5678"), add[1]),
   Person("Bob", 40, ["gardening", "cooking"], {"fiction": "The Great Gatsby", "non-fiction":
"Atomic Habits"}, ("bob@example.com", "555-9101"), add[2]),
]
```

Upewnij się, że obiekty mają atrybuty odpowiadające polom, które zdefiniowałeś.

4. Wywołaj metodę to_csv_file() na obiekcie csv_mapper, przekazując listę obiektów i nazwę pliku, do którego chcesz zapisać dane. Na przykład:

```
mapper1.to_csv_file(add, "address.csv")
mapper.to_csv_file(people, "people.csv")
```

5. W przypadku zagnieżdżeń obiektów klasy Address w obiektach klasy Person konieczna jest również dodanie **elif** do funkcji To_csv, na podstawie którego program będzie mógł zidentyfikować napotkany obiekt.

```
def to_csv(self, obj):
    row = []
    for field in self.fields:
        value = getattr(obj, field)
        if isinstance(value, list) or isinstance(value, set):
            value = ';'.join(map(str, value))
        elif isinstance(value, dict):
            value = ','.join([f"{k}:{v}" for k, v in value.items()])
        elif isinstance(value, tuple):
            value = ','.join(str(x) for x in value)

        elif isinstance(value, Address):

            value = "$" + value.a_id + "$"

        else:
            value = str(value)
        row.append(value)
    return row
```

6. Odczytywanie danych z plików odbywa się za pomocą funkcji **from_csv_file.**

```
people_from_csv = mapper.from_csv_file("people.csv")
addresses_from_csv = mapper1.from_csv_file("address.csv")
```

7. Należy pamiętać że w przypadku przechowywania zagnieżdżeń obiektu w obiekcie, po odczytaniu z pliku obiekt przechowuje znak specjalny (np. $) i numer identyfikacyjny drugiego obiektu. Do uzyskania poprawnego efektu należy użyć pętli która dopasuje obiekty po id.

```
for person in people_from_csv:
    for address in addresses_from_csv:
        if str( "$" + address.a_id +"$" ) == str(person.address):
            person.address = address
            break
```

W programie tworzone są obiekty klasy **Person** i przypisywane do listy **people**. Każdy obiekt Person jest inicjalizowany za pomocą argumentów, w których ostatni argument to obiekt klasy **Address** (np. add[0]). W ten sposób obiekt klasy **Person** przechowuje referencję do zagnieżdżonego obiektu klasy **Address**. Odczytywanie obiektów z plików w dostarczonym kodzie programu odbywa się za pomocą metody **from_csv_file** klasy **ObjectCSVMapper.** Metoda **from_csv_file** otwiera plik CSV i odczytuje jego zawartość przy użyciu obiektu **csv.reader**. Każdy wiersz z pliku CSV jest przekształcany z powrotem na obiekt za pomocą metody **from_csv**. Odczytane obiekty są następnie dodawane do listy **objs** i zwracane jako wynik metody **from_csv_file**.

Zawartość plików csv przechowujących obiekty:

people.csv

```
name,age,hobbies,favorite_books,contact_info,address
John,25,reading;hiking,"fiction:1984,non-fiction:The Art of Thinking Clearly","john@example.com,555-1234",$id_1$
Jane,30,swimming;dancing,"fiction:Pride and Prejudice,non-fiction:Sapiens","jane@example.com,555-5678",$id_2$
Bob,40,gardening;cooking,"fiction:The Great Gatsby,non-fiction:Atomic Habits","bob@example.com,555-9101",$id_3$
```

adress.csv

```
a_id,street,city
id_1,Wall st.,New York
id_2,Rodeo Drive,Los Angeles
id_3,Main st.,Dallas
```