|  |  |
| --- | --- |
| **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.

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):  value = ';'.join(value)  elif not isinstance(value, str):  value = str(value)  row.append(value)  return row    def from\_csv(self, row):  obj = type('', (), {})()  for i, field in enumerate(self.fields):  value = row[i]  if ';' in value:  value = value.split(';')  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)  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:  writer.writerow(self.to\_csv(obj))    def from\_csv\_file(self, filename):  objs = []  with open(filename, 'r', newline='') as csvfile:  reader = csv.reader(csvfile)  next(reader)  for row in reader:  obj = self.from\_csv(row)  objs.append(obj)  return objs |

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)**

Metoda inicjująca obiekt klasy ObjectCSVMapper.

Argumenty:

* fields - lista pól obiektu, które będą zapisane w pliku CSV. Każde pole musi być nazwą atrybutu obiektu.

**to\_csv(self, obj)**

Metoda zamieniająca obiekt Pythona na listę reprezentującą wiersz w pliku CSV.

Argumenty:

* obj - obiekt Pythona, który zostanie zamieniony na wiersz w pliku CSV.

Zwraca:

* Lista zawierająca wartości pól obiektu, które zostaną zapisane w pliku CSV.

**from\_csv(self, row)**

Metoda zamieniająca wiersz w pliku CSV na obiekt Pythona.

Argumenty:

* row - lista zawierająca wartości pól obiektu, które zostaną wczytane z pliku CSV.

Zwraca:

* Obiekt Pythona zawierający wartości pól wczytane z pliku CSV.

**to\_csv\_file(self, objs, filename)**

Metoda zapisująca listę obiektów do pliku CSV.

Argumenty:

* objs - lista obiektów Pythona, które zostaną zapisane w pliku CSV.
* filename - nazwa pliku, do którego zostaną zapisane dane.

**from\_csv\_file(self, filename)**

Metoda wczytująca listę obiektów z pliku CSV.

Argumenty:

* filename - nazwa pliku, z którego zostaną wczytane dane.

Zwraca:

* Lista obiektów Pythona wczytanych z pliku CSV.

Przykład użycia:

|  |
| --- |
| class Person:  def \_\_init\_\_(self, name, age, email):  self.name = name  self.age = age  self.email = email  fields = ['name', 'age', 'email']  mapper = ObjectCSVMapper(fields)  person1 = Person('John Smith', 30, 'john@example.com')  person2 = Person('Jane Doe', 25, 'jane@example.com')  persons = [person1, person2]  mapper.to\_csv(person1) # ['John Smith', 30, 'john@example.com']  mapper.from\_csv(['Jane Doe', '25', 'jane@example.com']) # <\_\_main\_\_.Person object at 0x7f8749f32080>  mapper.to\_csv\_file(persons, 'persons.csv')  loaded\_persons = mapper.from\_csv\_file('persons.csv') |

W tym przykładzie tworzymy klasę Person, która ma trzy pola: name, age i email. Następnie tworzymy instancje tej klasy i przechowujemy je na liście persons.

Tworzymy instancję ObjectCSVMapper, przekazując listę nazw pól, a następnie korzystamy z metod to\_csv, from\_csv, to\_csv\_file i from\_csv\_file do konwersji między obiektami a plikami CSV.

Na końcu ładowane są dane z pliku CSV i przechowywane w liście loaded\_persons, która zawiera wcześniej zapisane obiekty

Przykład użycia:

|  |
| --- |
| class Person:  def \_\_init\_\_(self, name, age, hobbies):  self.name = name  self.age = age  self.hobbies = hobbies    people = [  Person('Alice', 25, ['reading', 'hiking']),  Person('Bob', 30, ['gaming', 'cooking']),  Person('Charlie', 35, ['traveling'])  ]  mapper = ObjectCSVMapper(['name', 'age', 'hobbies'])  mapper.to\_csv\_file(people, 'people.csv')  loaded\_people = mapper.from\_csv\_file('people.csv')  for person in loaded\_people:  print(person.name, person.age, person.hobbies) |