Technologie obiektowe - projekt

Maciej Bandura, Marcin Ślusarczyk

12 czerwca 2024

1. Wstęp

Wybrany projekt: Object CSV Mapper, Object Spreadsheet Mapper
W części teoretycznej powinno znaleźć się omówienie odwzorowania pomiędzy wybranym językiem obiektowym oraz CSV i .xls, a w części praktycznej ich implementacja. Projekt w trakcie tworzenia został zmodyfikowany. W początkowej wersji zaprojektowano oraz zaimplementowano w pełni funkcjonalny mapper obiektowo-relacyjny na format CSV. W końcowej fazie projektu, postanowiono, zgodnie z ustaleniami poszerzyć wachlarz obsługiwanych formatów.

Obsługiwane formaty: .csv, .xlsx, .xml, .json. Źródłowa mechanika mappera oparta jest tylko i wyłącznie o format **.csv**, tzn. wszelkie obiekty mapowane są tylko z lub na ten format. Nie mniej jednak w trakcie rozwoju projektu powstał `ExtensionProvider`. Jest to strategia konwersji między-formatowej na etapie operacji IO (odczytu i zapisu). Pozwala to na zdefiniowanie własnego konwertera, dzięki któremu można przekształcić różne formaty z, oraz na .csv. W klasie `CSVMapper` domyślnie powołanym ExtensionProvider'em jest `CSVExtensionProvider`. Który de-facto nie dokonuje żadnej konwersji.

Język programowania: PHP
Wybrany ze względu na prostotę obsługi obiektów. Oraz *weak typing*.

Główne funkcjonalności mappera:

- Mapper posiada metodę pozwalającą na odczytanie pojedynczego obiektu z pliku (metoda `read()`).
- Mapper posiada możliwość odczytania kolekcji obiektów z pliku.
- Mapper umożliwia zapisanie obiektu do pliku (metoda `save()`).
- Mapper posiada możliwość zapisania kolekcji obiektów do pliku.
- Listy, tablice oraz inne kolekcje mapowane są dynamicznie z powiązanych plików.
- Mechanika mappera oparta jest o format .csv (oddzielany średnikiem).
- Definicja klasy jest wymagana do prawidłowego działania mappera.
- Instancja mappera wstrzykiwana jest do pól z odpowiednim dekoratorem (jak się to przyjęło robić w języku PHP).
- Klasy korzystające z dekoratora do instancjonowania mappera, muszą wykorzystywać trait [1] `CSVMapperInjector` oraz wywoływać pozyskaną metodę `injectDependencies()`.
- Każdy rodzaj klasy odwzorowywany jest w osobnym pliku (za wyjątkiem formatu .xlsx, gdzie obiekty tej samej klasy współdzielą jeden arkusz).

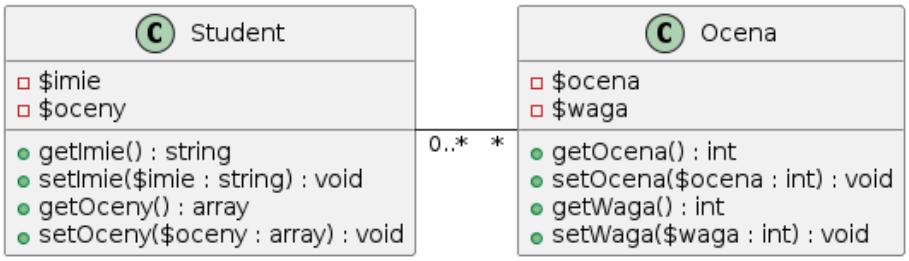
[1] https://www.php.net/manual/en/language.oop5.traits.php

2. Koncepcja mapowania obiektów na format CSV

Opracowana została struktura pliku .csv, umożliwiająca efektywne mapowanie obiektów, do tego formatu. Pozwala ona na zapis wartości prymitywnych, list takich wartości, obiektów oraz list obiektów. Każdy zapisany obiekt, otrzymuje losowy identyfikator `uniqueid()`[2], natomiast obiekty różnych klas zapisywane są w różnych plikach. Nazwy tych plików odpowiadają przestrzenią nazw tych klas w PHP.

W pierwszej linijce pliku .csv - nagłówku, przechowujemy informacje opisujące dane, są to, między innymi: nazwy pól w mappowanej klasie, wartości prefiksowane przy pomocy znaku "~"informują nas o listach, natomiast wartości zawierające znak "@"wskazują na referencje do obiektów (np. obiektów innej klasy). Dodatkowemu polu **id**, przypisano znak specjalny: "#", tak aby nie kolidował z rzeczywistym polem *id*, które często jest używane w obiektach. Znaki te zostały wybrane, ze względu na brak możliwości wystąpienia ich w rzeczywistej nazwie pola w języku PHP. Pozostała zawartość pliku csv nazywana przez nas *ciałem*, zawiera rzeczywiste mapowanie do obiektów, opisane według definicji z nagłówka.

Przykład mappowania: Klasa Student i Ocena



Rysunek 1: Mapowane klasy

```
1 imie; ~oceny@TestListsObjs\Ocena; id#
2 Adam; 6664a3116c12c,6664a3116c131, ... ,6664a3116c13d; 6664a3116c124
```

Listing 1: Plik TestListsObjs-Student.csv

```
1 ocena; waga; id#
2 6; 1; 6664a3116c12c
3 5; 2; 6664a3116c131
4 ...
5 1; 6; 6664a3116c13d
```

Listing 2: Plik TestListsObjs-Ocena.csv

[2]https://www.php.net/manual/en/function.uniqid.php

3. Wykorzystanie

Jak zostało wspomniane we wstępie, aby użyć mappera należy, wykorzystywać trait `CSVMapperInjector` oraz wywoływać pozyskaną metodę `injectDependencies()`.

```php
<?php
  require 'fake_vendor/autoload.php';
  require 'vendor/autoload.php';
  use CSVMapper\Boostrapper\CSVMapperInjector;

  class App
  {
    use CSVMapperInjector;

    public function __construct ()
    {
      $this->injectDependencies();
    }

    /**
    * @CSVMapper
    */
    private $csvMapper;

    public function main ()
    {
      var_dump($this->csvMapper);
    }

  }

  (new App())->main();
```

Listing 3: Przykład powołania mappera.

Mając już obiekt, `csvMappera`, możemy dokonywać operacji mapowania obiektów z pliku, jak i również ich zapisywania. Przy pomocy metod `read()` oraz `save()`.

```php
public function main ()
{
  $myClass = new MyClass();
  $this->csvMapper->save($myClass);
}
```

Listing 4: Przykład zapisu obiektu klasy **MyClass** do pliku.

```php
public function main ()
{
  $myObject = $this
    ->csvMapper
    ->read("./MyClass.csv", MyClass::class);

  var_dump($myObject);
}
```

Listing 5: Przykład odczytu obiektu klasy **MyClass** z pliku.

4. Testy funkcjonalności

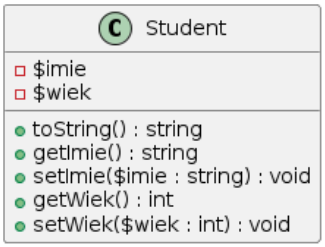
W celu zachowania poprawności działania zostały przygotowane testy mające na celu sprawdzanie działania mappera. Testy opiewały o podstawowe funkcjonalności jak i te bardziej złożone. Za wykonywanie testów odpowiedzialny jest plik `tests.php`, zawiera on główną klasę `Tests`, która w konstruktorze powołuje konkretne testy rozszerzające klasę `Test`. Wszystkie testowane klasy zawierają metodę `test()`, w której zawarta jest logika testu. Metoda ta, musi zwrócić wartość wywołania `pass()` - jeżeli test się powiedzie, lub `fail()` - jeżeli dojdzie to jakiegoś zdarzenia. Metody te odpowiednio obsłużą raportowanie wyników testu.

```
archpad :: ~/sem1/technologie_obiektowe_projekt » php ./tests.php
[PASS] Serializacja Obiektów (Zapis/Odczyt)
[PASS] Serializacja i mapowanie typów pól w obiektach
[PASS] Kopie obiektów
[PASS] Relacje cykliczne
[PASS] Serializacja list
[PASS] Serializacja list obiektów
[PASS] Dziedziczenie
[PASS] Kompozycja
Test Count: 8, Passed: 8, Failed: 0
```

Rysunek 2: Raport wykonanych testów

4.1 Test serializacji obiektów

Jest to najbardziej podstawowy test, mają5cy na celu weryfikację działania mappera pod kątem zapisu i odczytu prostych obiektów, niezawierających list ani referencji do innych obiektów.



Rysunek 3: Serializowana klasa **student**

```
1 imie; wiek; id#
2 Tomek; 18; 666613345af2d
```

Listing 6: Wygenerowany plik TestSerializacjiObiektow-Student.csv

```php
public function test ()
{
  if ($this->mapper == null) {
    return $this->fail();
  }

  $student = new Student();
  $student->setWiek(18);
  $student->setImie("Tomek");

  $this->mapper->save($student);
  $fromFile = $this->mapper->read("./TestSerializacjiObiektow\Student.csv",
  ↪  Student::class);

  if (get_class($fromFile) != Student::class) {
    return $this->fail();
  }

  if ($student->toString() != $fromFile->toString()) {
    return $this->fail();
  }

  return $this->pass();
}
```

Listing 7: Logika testu.

4.2 Test serializacji typów

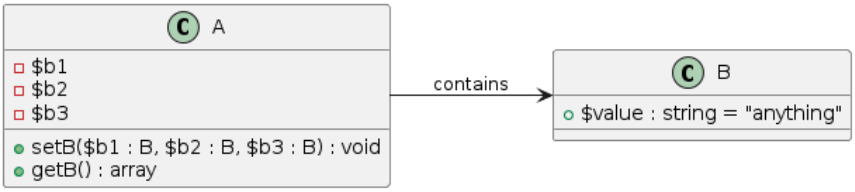
Kolejny, również prosty test, mający na celu sprawdzenie poprawności serializacji typów pól. Po zapisie sprawdzane jest, czy typy pól odczytanego obiektu zgadzają się z typami oryginalnego obiektu.

```php
if (gettype($fromFile->getWysokosc()) != gettype($student->getWysokosc())) {
  return $this->fail();
}
if (gettype($fromFile->getWiek()) != gettype($student->getWiek())) {
  return $this->fail();
}
if (gettype($fromFile->getImie()) != gettype($student->getImie())) {
  return $this->fail();
}
```

Listing 8: Logika testu.

4.3 Test kopii obiektów

Test ma na celu zweryfikować, poprawność serializacji wielu referencji do tego samego obiektu, w obiekcie **A**, pola `$b1` oraz `$b2` zawierają referencje do tego samego obiektu. Natomiast pole `$b3` zawiera referencje do innego obiektu tej samej klasy. Test upewnia się, że zostaną zserializowane tylko dwie instancje klasy **B**, oraz to, że zostaną one poprawnie zmapowane po odczytaniu.



Rysunek 4: Serializowane klasy

```
b1@TestObjectCopies\B; b2@TestObjectCopies\B; b3@TestObjectCopies\B; id#
666619324645f; 666619324645f; 6666193246466; 6666193246459
```

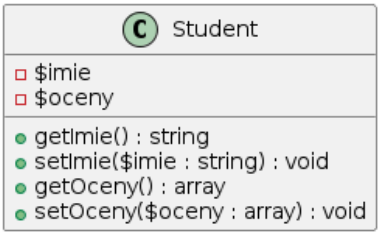
Listing 9: Plik TestObjectCopies-A.csv

```
value; id#
anything; 666619324645f
anything; 6666193246466
```

Listing 10: Plik TestObjectCopies-B.csv

4.4 Test serializacji list

Prosty test serializacji list typów prymitywnych.



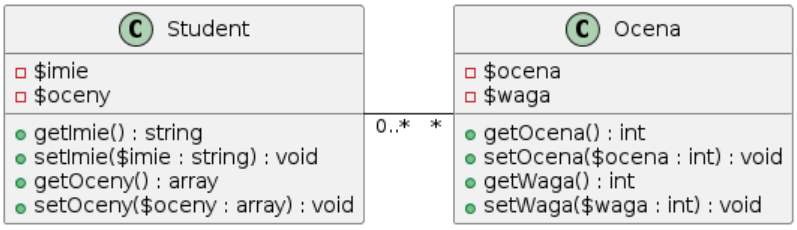
Rysunek 5: Mapowana klasa

```
imie; ~oceny; id#
Szymek; 2,2,3,1; 66661932466e6
```

Listing 11: Plik TestLists-Student.csv

4.5 Test serializacji list obiektów

Jest to rozszerzenie poprzedniego testu o referencje do obiektów innej klasy. Idea testu pozostaje taka sama.

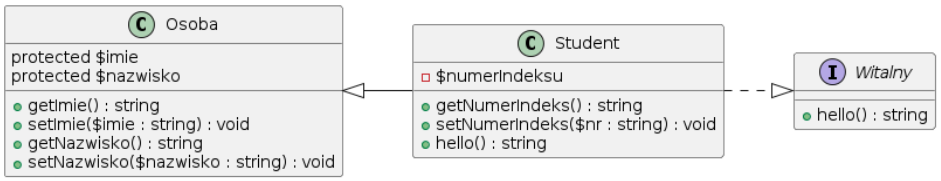


Rysunek 6: Mapowana klasa

Struktura plików .csv została zaprezentowana we wstępie.

4.6 Test - dziedziczenie

W programowaniu obiektowym jednym z podstawowych narzędzie jest dziedziczenie oraz implementowanie. Oczywistym jest, że nasz mapper musi współpracować z obiektami posługującymi się tymi narzędziami. Test ten ma na celu weryfikację, czy odtworzony obiekt odpowiednio dziedziczy po klasie `Osoba` oraz czy implementuje interfejs `Witalny`.



Rysunek 7: Mapowana klasa

```
1  if (!($fromFile instanceof Student)) {
2    return $this->fail();
3  }
4
5  if (!($fromFile instanceof Osoba)) {
6    return $this->fail();
7  }
8
9  if (!($fromFile instanceof Witalny)) {
10   return $this->fail();
11 }
```

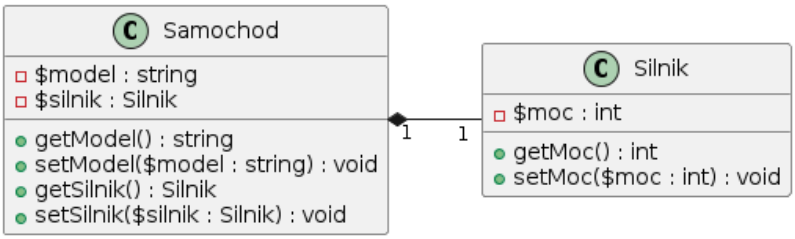
Listing 12: Logika testu

```
numerIndeksu; imie; nazwisko; id#
abc123; Kamil; Ślimak; 6666193246903
```

Listing 13: Plik TestDziedziczenia-Student.csv

4.7 Test - kompozycja

Kolejnym narzędziem wykorzystywanym w technologiach obiektowych jest kompozycja. Poniższy test, zapewnia, pełną obsługę kompozycji przez naszego mappera.



Rysunek 8: Mapowana klasa

```php
if ($fromFile->getModel() != $bmw->getModel()) {
  return $this->fail();
}

if (!($fromFile->getSilnik() instanceof Silnik)) {
  return $this->fail();
}

if ($fromFile->getSilnik()->getMoc() != $silnik->getMoc()) {
  return $this->fail();
}
```

Listing 14: Logika testu

```
model; silnik@TestKompozycja\Silnik; id#
528i; 66661932469af; 66661932469a8
```

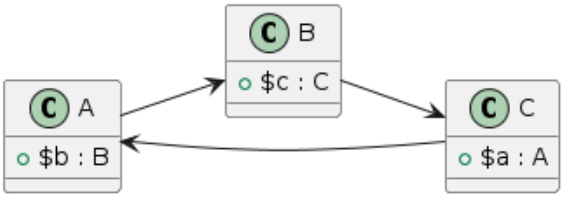
Listing 15: Plik TestKompozycja-Samochod.csv

```
moc; id#
245KM; 66661932469af
```

Listing 16: Plik TestKompozycja-Silnik.csv

4.8 Test relacji cyklicznych

Relacje cykliczne zwane również rekurencyjnymi są ciekawym zjawiskiem, do których często dochodzi przez przypadek w środowiskach produkcyjnych. Niezależnie od tego, czy są dobrymi praktykami, czy nie, nasz mapper musi je wspierać. Ważne jest aby w trakcie odtwarzania obiektów, nie dopuścić do zapętlenia mappera. Takie zjawisko można wykryć przy pomocy tego testu.



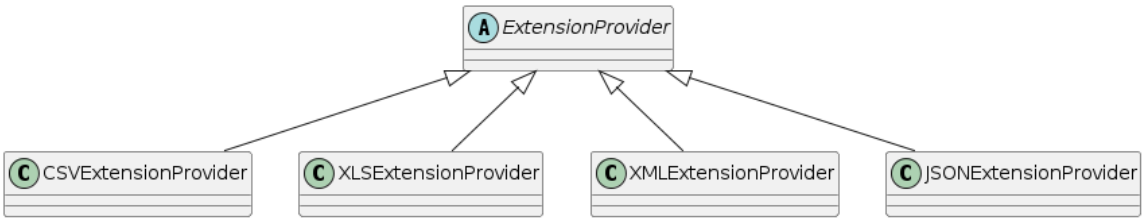
Rysunek 9: Mapowana klasa

```php
if ($fromFile->b->c->a !== $fromFile) {
  return $this->fail();
}
```

Listing 17: Logika testu

5. Mappery na inne formaty

Jak zostało wspomniane we wstępie, nasz mapper jest w stanie obsługiwać różne formaty, nie tylko .csv. Nie mniej jednak nie posiada on logiki do dokładnego mapowania z każdego z obsługiwanych formatów. Posiada on serię konwerterów między formatowych pozwalających tłumaczyć różne formaty na csv i na odwrót. Mechanizm działania samego mappera współpracuje tylko i wyłącznie z formatem csv, konwertery zapewniają dodatkową warstwę wsparcia poszerzając wachlarz wspieranych formatów. Konwersja ta, realizowana jest przy pomocy mechaniki `ExtensionProvider`, gdzie każdy konkretny "*dostawca rozszerzeń*"funkcjonuje jako swojego rodzaju tłumacz między rdzeniem mappera a konkretnymi formatami.



Rysunek 10: Dostępni dostawcy formatów/rozszerzeń

Konkretnego dostawcę możemy wstrzyknąć do instancji mappera, przy pomocy metody `provideExtension(ExtensionProvider)`, nadpisując w ten sposób domyślnego dostawcę, tj. `CSVExtensionProvider`.

```
1  public function main ()
2  {
3    $this->csvMapper
4    ->provideExtension(new XMLExtensionProvider())
5    ->save($this->a);
6
7    $x = $this->csvMapper
8    ->read("./A.csv", A::class);
9  }
```

Listing 18: Wstrzyknięcie dostawcy formatu XML

Uwaga: Mimo, że jako ścieżkę podaliśmy plik z rozszerzeniem *.csv*, to dostawca doklei do niego rozszerzenie *.xml*.

5.1 CSVExtensionProvider

Jest to domyślny oraz najprostszy dostawca, którego zadaniem jest tylko zapisać i odczytać podany plik.

```php
namespace CSVMapper\ExtensionProvider;
use CSVMapper\ExtensionProvider\ExtensionProvider;

class CSVExtensionProvider implements ExtensionProvider
{
  public function write ($file, $csv)
  {
    file_put_contents($file, $csv);
  }

  public function read ($file)
  {
    return file_get_contents($file);
  }
}
```

Listing 19: CSVExtensionProvider

5.2 XMLExtensionProvider

Nieco bardziej zaawansowanym dostawcą jest, ten od formatu XML. Wykorzystując wbudowaną w język PHP bibliotekę `DOMDocument`, odpowiednio tworzy i parsuje pliki .xml na podstawie dostarczanych danych w formacie CSV.

```xml
<?xml version="1.0"?>
<Mapping>
  <Definition>
    <Field>moc</Field>
    <Field>id#</Field>
  </Definition>
  <Content>
    <Entity>
      <Value>245KM</Value>
      <Value> 66662740bf785</Value>
    </Entity>
  </Content>
</Mapping>
```

Listing 20: TestKompozycja-Silnik.csv.xml

```xml
<?xml version="1.0"?>
<Mapping>
  <Definition>
    <Field>model</Field>
    <Field> silnik@TestKompozycja\Silnik</Field>
    <Field>id#</Field>
  </Definition>
  <Content>
    <Entity>
      <Value>528i</Value>
      <Value> 66662740bf785</Value>
      <Value> 66662740bf77f</Value>
    </Entity>
  </Content>
</Mapping>
```

Listing 21: TestKompozycja-Samochod.csv.xml

```php
public function write ($file, $csv)
{
  $lines = explode("\n", $csv);
  $header = explode(";", array_shift($lines));

  $domDoc = new DOMDocument;
  $root = $domDoc->createElement('Mapping');
  ...
  foreach ($header as $tok) {
    /**
    * Konwersja nagłowka ...
    */
  }

  $content = $domDoc->createElement('Content');
  $root->appendChild($content);
  foreach ($lines as $line) {
    /**
    * Konwersja ciała ...
    */
  }

  file_put_contents($file . ".xml", $domDoc->saveXML());
}
```

Listing 22: Fragment metody zapisu

5.3 JSONExtensionProvider

JSON jest jednym z najbardziej popularnych formatów, wymiany danych. Język PHP zawiera wbudowane funkcje do obsługi tego formatu, dlatego też, stworzenie konwertera nie stanowiło dużego wyzwania.

```php
public function read ($file)
{
  $json = file_get_contents($file . ".json");
  $root = json_decode($json);

  $header = [];
  $entities = [];

  $first = (array) $root->entities[0];
  foreach ($first as $key => $value) {
    $header[] = $key;
  }

  foreach ($root->entities as $entity) {
    $values = [];
    foreach ((array) $entity as $value) {
      $values[] = $value;
    }
    $entities[] = implode(";", $values);
  }

  return implode(";", $header) . "\n" . implode("\n", $entities);
}
```

Listing 23: Metoda odczytu

```json
{
  "entities":[
    {
      "model":"528i",
      " silnik@TestKompozycja\\Silnik":" 666629a031291",
      " id#":" 666629a031289"
    }
  ]
}
```

Listing 24: TestKompozycja-Samochod.csv.json

```
{
  "entities":[
    {
      "moc":"245KM",
      " id#":" 666629a031291"
    }
  ]
}
```

Listing 25: TestKompozycja-Silnik.csv.json

```
    public function write ($file, $csv)
    {
      $lines = explode("\n", $csv);
      $header = explode(";", array_shift($lines));

      $root = (object) ["entities" => []];

      foreach ($lines as $line) {
        $toks = explode(";", $line);
        $newObj = [];
        for ($i = 0; $i < count($toks); $i++) {
          $newObj[$header[$i]] = $toks[$i];
        }
        $root->entities[] = (object) $newObj;
      }

      file_put_contents($file . ".json", json_encode($root));
    }
```

Listing 26: Metoda zapisu

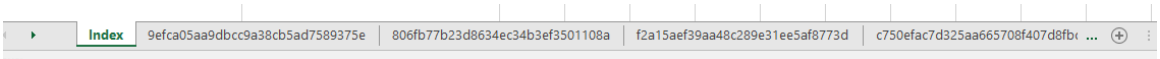
5.4 XLSExtensionProvider

Ostatnim z obsługiwanych formatów jest XLS. W przypadku poprzednich formatów każdy rodzaj klasy był zapisany w osobnym pliku. W tym przypadku zostało zastosowane inne podejście - obiekty tych samych klas zapisywane są we wspólnych *arkuszach*. Takie podejście wiąże się, z dodatkowymi komplikacjami. Format XLS posiada rygorystyczne ograniczenia dotyczące nazw arkuszy, dlatego też wszystkie nazwy musimy nadpisywać, tak aby spełniały one wspomniane kryteria. Dodatkowym problemem jest maksymalna długość nazwy, a mianowicie, jest to 31 znaków. Dlatego też został zastosowany manewr "numerowania"arkuszy. Ponieważ, osoba analizująca arkusz,

własnoręcznie może mieć problemy z transkrypcją numerów na nazwy plików, do każdego zmapowanego pliku *.xlsx*, dołączamy dodatkowy arkusz zawierający indeks nazw i odpowiadających im numerów. Ponadto język PHP natywnie nie wspiera formatu XLS, wymagana jest instalacja dodatkowej biblioteki `phpoffice/phpspreadsheet`. Aby tego dokonać, należy użyć narzędzia **composer** wywołując poniższe polecenie:
`# composer require phpoffice/phpspreadsheet`
Biblioteka ta, dodatkowo może wymagać konfiguracji ustawień samego PHP. Należy upewnić się, że poniższe moduły są włączone:
`extension=gd`
`extension=iconv`

	A	B
1		
2	9efca05aa9dbcc9a38cb5ad7589375e	./TestSerializacjiObiektow-Student.csv
3	806fb77b23d8634ec34b3ef3501108a	./TestTypes-Student.csv
4	f2a15aef39aa48c289e31ee5af8773d	./TestObjectCopies-B.csv
5	c750efac7d325aa665708f407d8fbc2	./TestObjectCopies-A.csv
6	8f61ef20650c1716366f7e03533b329	./TestCyclicRelations-C.csv
7	866c414e9613636f23fbfffe59110c8	./TestCyclicRelations-B.csv
8	8b44d8a9229391491a00fad35330512	./TestCyclicRelations-A.csv
9	316eff8ca8b4fad2ccc14b88612d44e	./TestLists-Student.csv
10	a3d727e7f89383623482b9062cd5bdf	./TestListsObjs-Ocena.csv
11	5e82e759bfb391695af6697ac5b0ef1	./TestListsObjs-Student.csv
12	cea2d265497ee41ef6c56079c613cf1	./TestDziedziczenia-Student.csv
13	be10686d3f0858d36031bad251051c5	./TestKompozycja-Silnik.csv
14	579d5971a9cbb83b41d70ea368ed033	./TestKompozycja-Samochod.csv

Rysunek 11: Indeks nazw arkuszy



Rysunek 12: Widok dostępnych arkuszy w pliku

	A	B	C
1	model	silnik@TestKompozycja\Silnik	id#
2	528i	6664bcc64fd42	6664bcc64fd3a

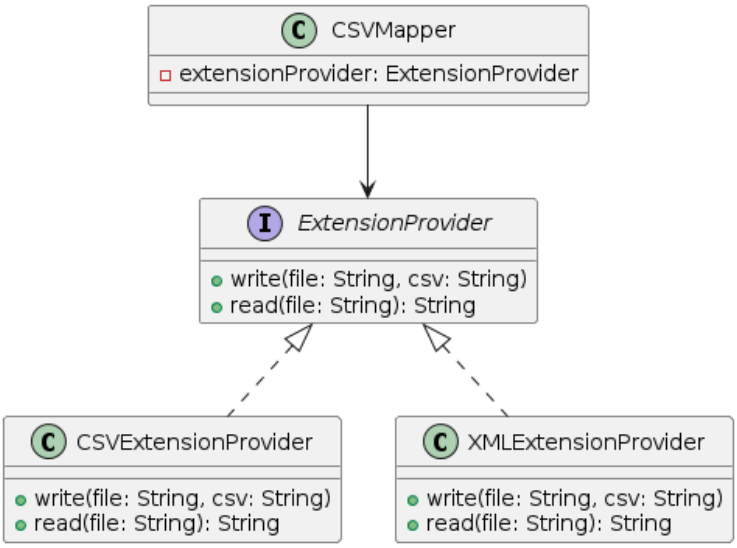
Rysunek 13: Arkusz TestKompozycja-Samochod.csv

6. Wykorzystane wzorce projektowe

6.1 Strategia

Wzorzec projektowy strategia to behawioralny wzorzec projektowy, który umożliwia definiowanie rodziny algorytmów, enkapsulowanie każdego z nich oraz uczynienie ich wymienialnymi. Wzorzec ten pozwala na zmienianie algorytmów niezależnie od klientów, które z nich korzystają. Główne elementy wzorca Strategia to: Kontekst (Context), Strategia (Strategy) oraz Konkretne Strategie (Concrete Strategies). Kontekst to klasa, która zawiera referencję do obiektu Strategii i deleguje do niego wykonanie pewnych operacji. Strategia to interfejs wspólny dla wszystkich algorytmów, który definiuje metodę, jaką muszą implementować wszystkie konkretne strategie. Konkretne Strategie to klasy implementujące interfejs Strategii, zawierające konkretne algorytmy. Wzorzec Strategia jest użyteczny, gdy istnieje potrzeba dynamicznego wyboru algorytmu w trakcie działania programu lub gdy chcemy uniknąć umieszczania wielu złożonych warunków w kodzie. Dzięki temu wzorcowi kod staje się bardziej elastyczny i łatwiejszy do utrzymania.

W przypadku naszego projektu, użyliśmy tego wzorca do implementacji dostawców formatów (konwerterów)



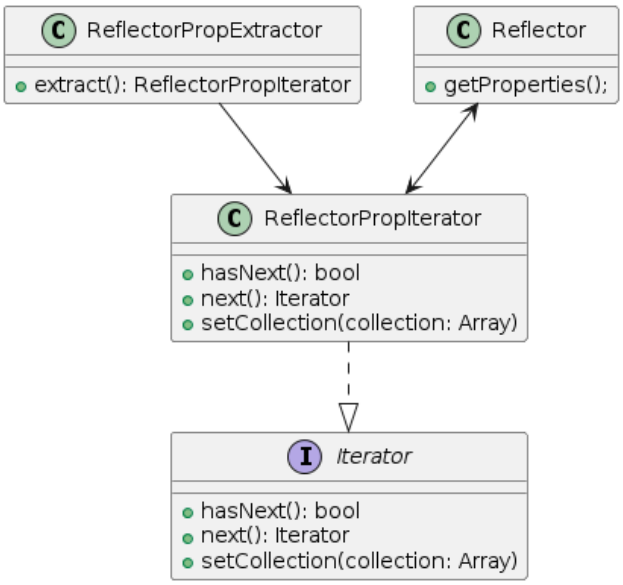
Rysunek 14: Wykorzystanie strategii

6.2 Iterator

Iterator to behawioralny wzorzec projektowy, który umożliwia sekwencyjne przechodzenie przez elementy kolekcji bez ujawniania jej wewnętrznej struktury (takiej jak lista, stos, drzewo, itp.). Dzięki temu wzorcowi można uzyskać jednolity interfejs do przeglądania różnych typów kolekcji, co ułatwia manipulację i przetwarzanie danych. Główne elementy wzorca Iterator to: Iterator, Kolekcja (Collection) oraz Konkretne

Iteratory (Concrete Iterators). Iterator to interfejs definiujący metody do iterowania po elementach kolekcji, takie jak hasNext() (sprawdzająca, czy są jeszcze elementy do przejścia) oraz next() (zwracająca kolejny element). Konkretne Iteratory to klasy implementujące interfejs Iteratora, które zawierają logikę potrzebną do przechodzenia po konkretnej strukturze danych.

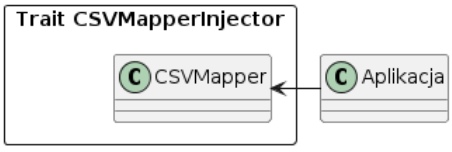
W przypadku naszego projektu, użyliśmy tego wzorca do implementacji wykrywania adnotacji w klasach wykorzystujących mappera.



Rysunek 15: Wykorzystanie iteratora

6.3 Dekoratror

Dekorator to strukturalny wzorzec projektowy, który pozwala dynamicznie dodawać nowe obowiązki obiektom poprzez umieszczanie tych obiektów w specjalnych obiektach opakowujących, które zawierają odpowiednie zachowania. Dzięki temu wzorcowi można rozszerzać funkcjonalność obiektów bez modyfikowania ich kodu bazowego. W kontekście PHP, dekorator może być również implementowany za pomocą cech (traits). Traits w PHP to mechanizm pozwalający na wielokrotne wykorzystanie kodu w różnych klasach.



Rysunek 16: Wykorzystanie dekoratora

7. Podsumowanie

Projekt można uznać za ukończony. Wszystkie założone funkcjonalności zostały zaprojektowanie i zaimplementowane. Mapper sprostał coraz to nowym wymaganiom stawianym w trakcie trwania semestru. Testy były ważnym narzędziem w trakcie prac nad projektem, nie tylko sprawdzały poprawność wprowadzanych funkcjonalności, ale również nakreślały drogę rozwoju projektu. Dlatego też złożoność testów jest różna, od tych prostszych po te bardziej zaawansowane.

Projekt współpracuje z wieloma formatami, gdzie każdy z nich został sprawdzony przy pomocy przedstawionego zestawu testów. Obsługę wielu formatów można było rozwiązać na dwa główne sposoby:

Pierwszy: dla każdego formatu można było stworzyć dedykowany mapper, wymagałoby to znacznych zmian w istniejącym kodzie. Tak naprawdę to należałoby rozdzielić istniejącą mechanikę na dwie, obsługę plików *.csv* oraz klasę do zarządzania mapowaniem ogólnie. Wiązałoby się to z bardzo dużym nakładem pracy, natomiast sam projekt znacznie odstawałby od swojej pierworodnej idei.

Drugi: można było stworzyć mechanizm konwerterów, współpracujący, z już gotową mechaniką mappera CSV - tak jak to zostało robione. Zastosowane podejście, zdało test, ma ono swoje plusy ale również minusy.

Plusy:

- Stosunkowo mała ilość zmian w istniejącym kodzie.
- Bardzo szybki development nowych dostawców formatów.
- Rozdzielona odpowiedzialność między konwerterami a mapperem.
- Nie ogranicza nas abstrakcja danych w formacie, z perspektywy mappera.

Minusy:

- Niska wydajność w porównaniu do dedykowanego mappera konkretnego formatu
- Wszelkie niedociągnięcia następujące w trakcie konwersji.