Projektowanie obiektowe

Laboratorium 4
Refaktoryzacja

Kwiecień 2020

1 Opis

Celem laboratorium jest poznanie metod ulepszenia struktury kodu źródłowego w celu jego łatwiejszej rozbudowy lub modyfikacji z wykorzystaniem diagramu klas. Celem dodatkowym jest użycie wzorca behawioralnego Iterator.

2 Przygotowanie do zajęć

Pobierz kod, który jest dołączony do laboratorium. Uruchom projekt w środowisku IntelliJ. Zapoznaj się ze strukturą kodu. Kod jest celowo zbudowany w sposób nieoptymalny.

3 Opis aplikacji

Aplikacja imituje prostą wyszukiwarkę osób, która ma pomagać służbom w odnajdywaniu obywateli, którzy mogli dopuścić się przestępstwa. Przykładowe funkcje to:

1. wyszukiwanie po imieniu lub nazwisku osoby,
2. wyszukiwanie więźniów z całej Polski,
3. wyszukiwanie obywateli Krakowa.

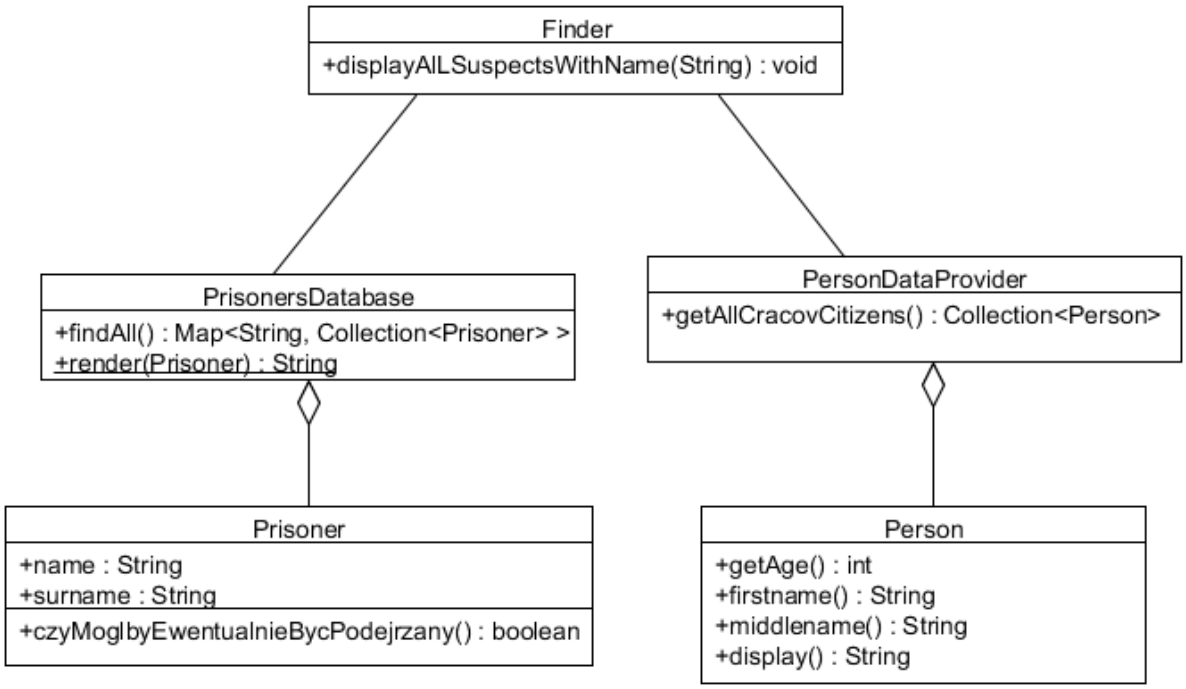
Planowane nowe funkcje:

1. wyszukiwanie po wieku osoby,
2. obsługa nowego zbioru z danymi o krakowskich studentach.

4 Zadania

4.1 Krok 1

Narysuj diagram UML na podstawie analizy kodu. Porównaj otrzymany diagram z poniższym. Zaproponuj zmiany poprawiające jakość i czytelność kodu źródłowego (np. nazewnictwo i widoczność metod, hierarchia klas).



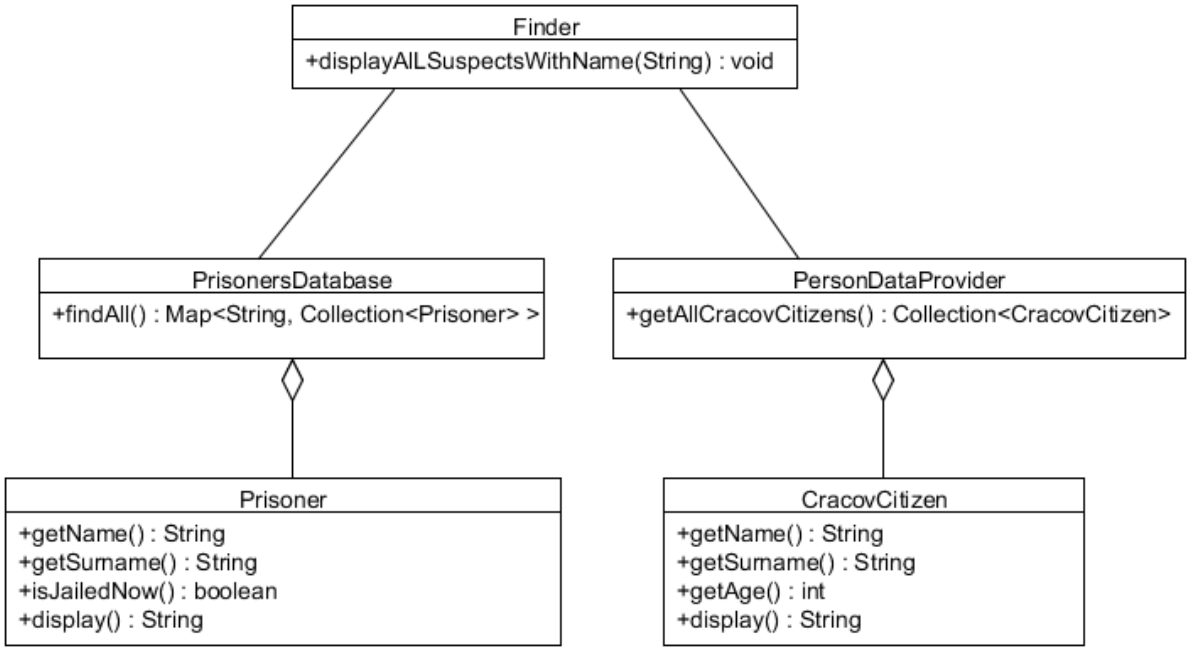
Rysunek 1: Diagram ilustrujący oryginalną strukturę kodu.

4.2 Krok 2

Wprowadź poprawę podstawowych błędów w kodzie, takich jak:

1. publiczne pola zamiast metod dostępowych,
2. statyczne metody w złych miejscach,
3. zbyt długie i enigmatyczne nazwy metod.

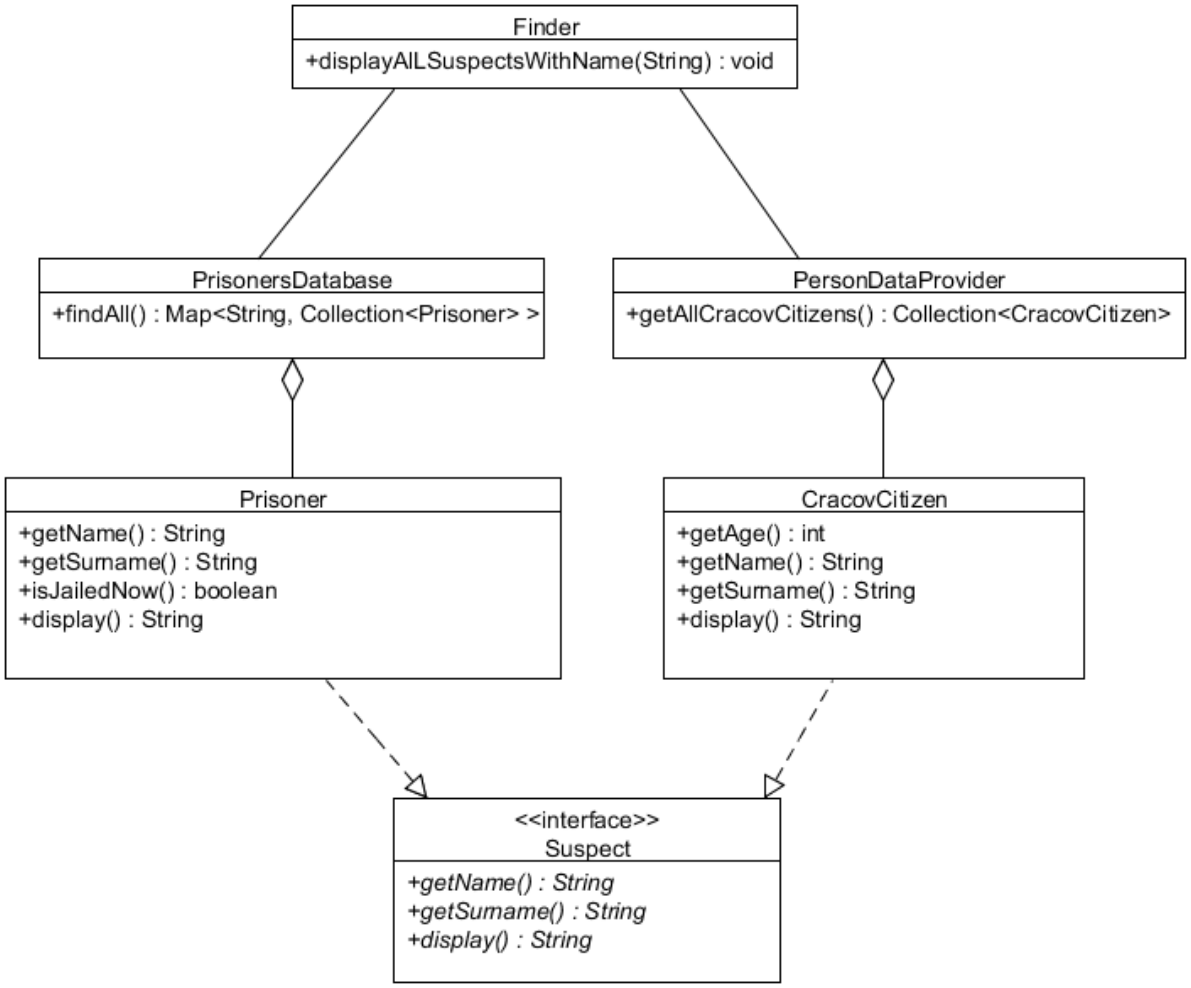
Skorzystaj z narzędzi do refaktoryzacji udostępnianych przez środowisko programistyczne. Propozycję poprawionej struktury przedstawia Diagram 2.



Rysunek 2: Propozycja poprawionej struktury kodu.

4.3 Krok 3

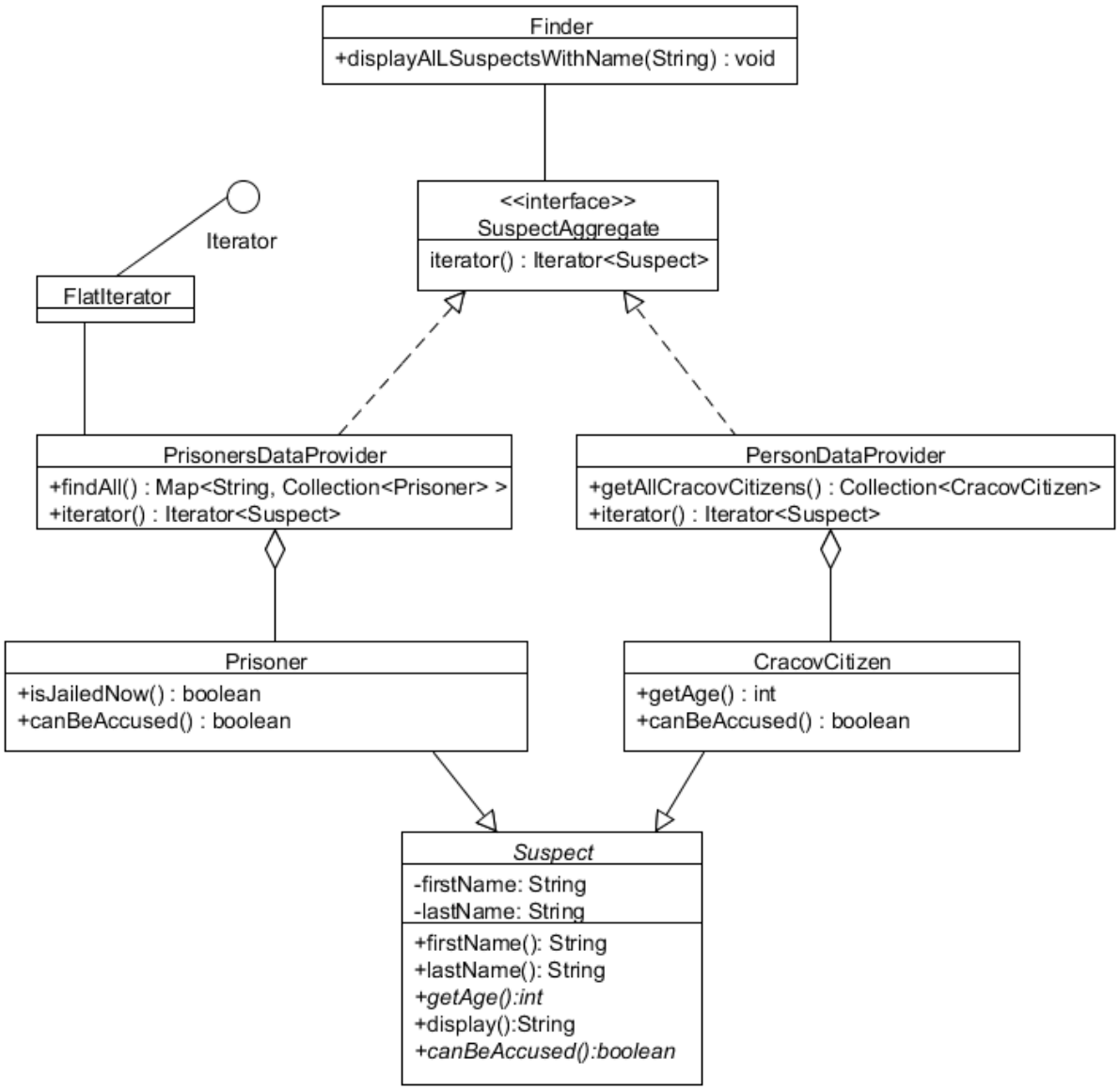
Zaproponuj generalizację klas `Person` i `Prisoner`. Uzasadnij użycie inferfejsu, klasy abstrakcyjnej lub interfejsu z metodami domyślnymi (Java 8). Propozycję zmienionej struktury przedstawia Diagram 3. Zastanów się jakie metody można dodać do klasy `Suspect` aby uogólnić klasę `Finder`.



Rysunek 3: Propozycja poprawionej struktury kodu.

4.4 Krok 4

Kolejnym krokiem jest wprowadzenie generalizacji do klas dostarczających dane do systemu. Ważne jest w tym przypadku, że nie mamy wpływu na to jak one wyglądają (przyjmujemy, że implementacje tych klas są dostarczane z zewnątrz). Nie możemy więc zmienić metod występujących w ich interfejsach. Możemy jedynie metody dodać lub (najlepiej) udekorować wzorcem `Dekorator`.
Dodatkowym problemem jest odmienna struktura danych zwracana przez obu dostawców. Jeden zwraca `Collection<CracovCitizen>`, drugi zaś `Map<String, Collection<Prisoner»`. Ponadto, ponieważ zbiory danych są duże, nie chcemy tworzyć nowej kolekcji tylko po to by ją zwrócić. Rozwiązaniem może być `Iterator`. W przypadku `PersonDataProvider` dostarcza go interfejs `Collection`. Dla `PrisonerDatabase` musimy zaimplementować swój iterator, implementujący interfejs `Iterator` z Javy, który dostarczy interesujące nas elementy w tej samej formie. Propozycję rozszerzonej struktury przedstawia Diagram 4.

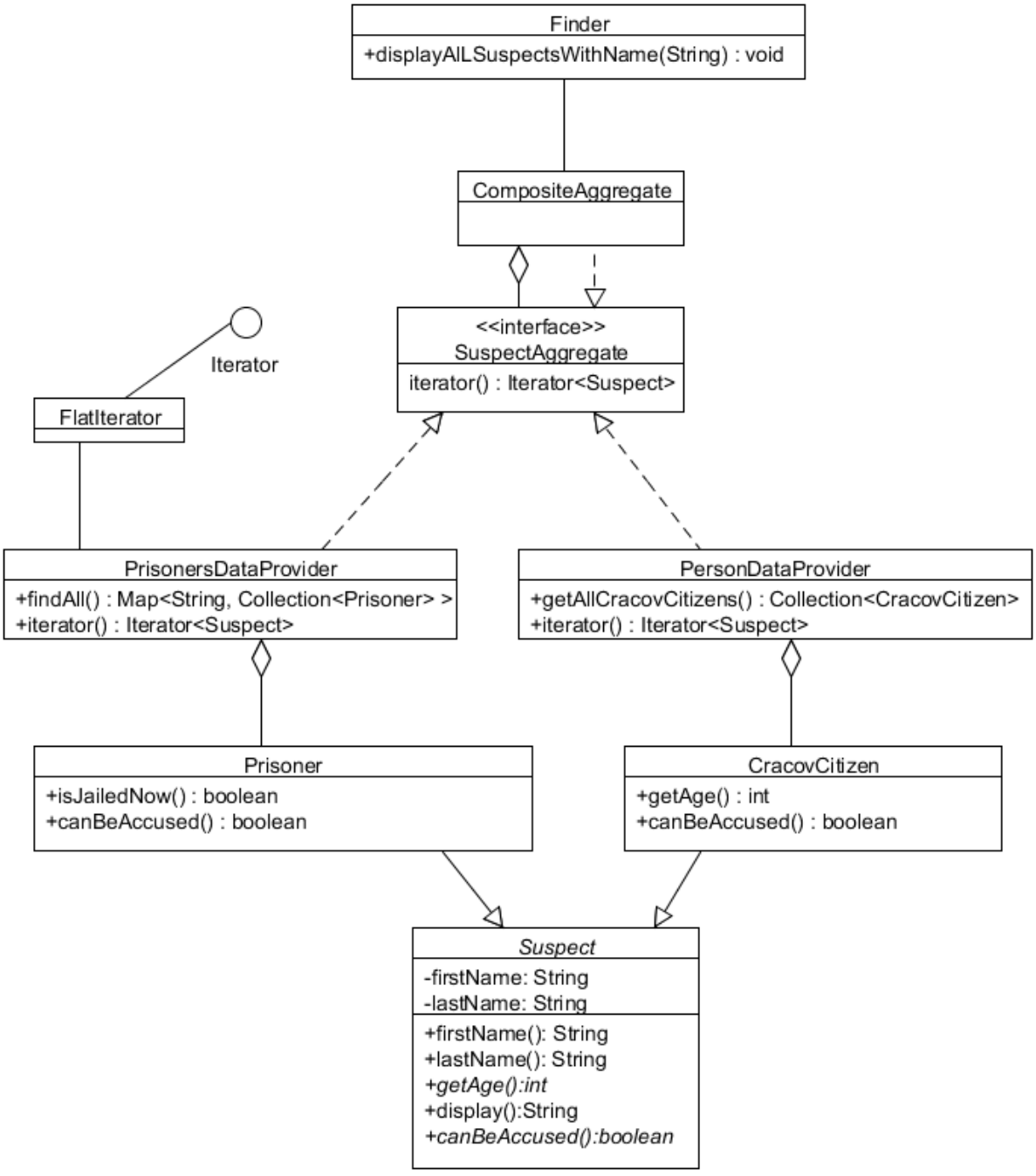


Rysunek 4: Propozycja rozbudowanej struktury kodu.

4.5 Krok 5

Czy problem dodania nowego zbioru danych został rozwiązany? Czy wystarczy dostosować nową implementację do `SuspectAggregate` i wszystko będzie działać? Niestety, `Finder` nadal musi wiedzieć, ile jest zbiorów danych, więc trzeba zmienić jego implementację, aby go nie modyfikować przy nowych zbiorach danych. Obecnie, po tym że `Finder` jest połączony bezpośrednio z instancjami `SuspectAggregate` widać, że wewnątrz ciała klasy `Finder` są zagnieżdżone pętle.
Należy więc dodać klasę pośrednią pomiędzy agregatami a finderem, której odpowiedzialnością będzie agregacja wszystkich osób ze wszystkich zbiorów danych i przekazanie ich do

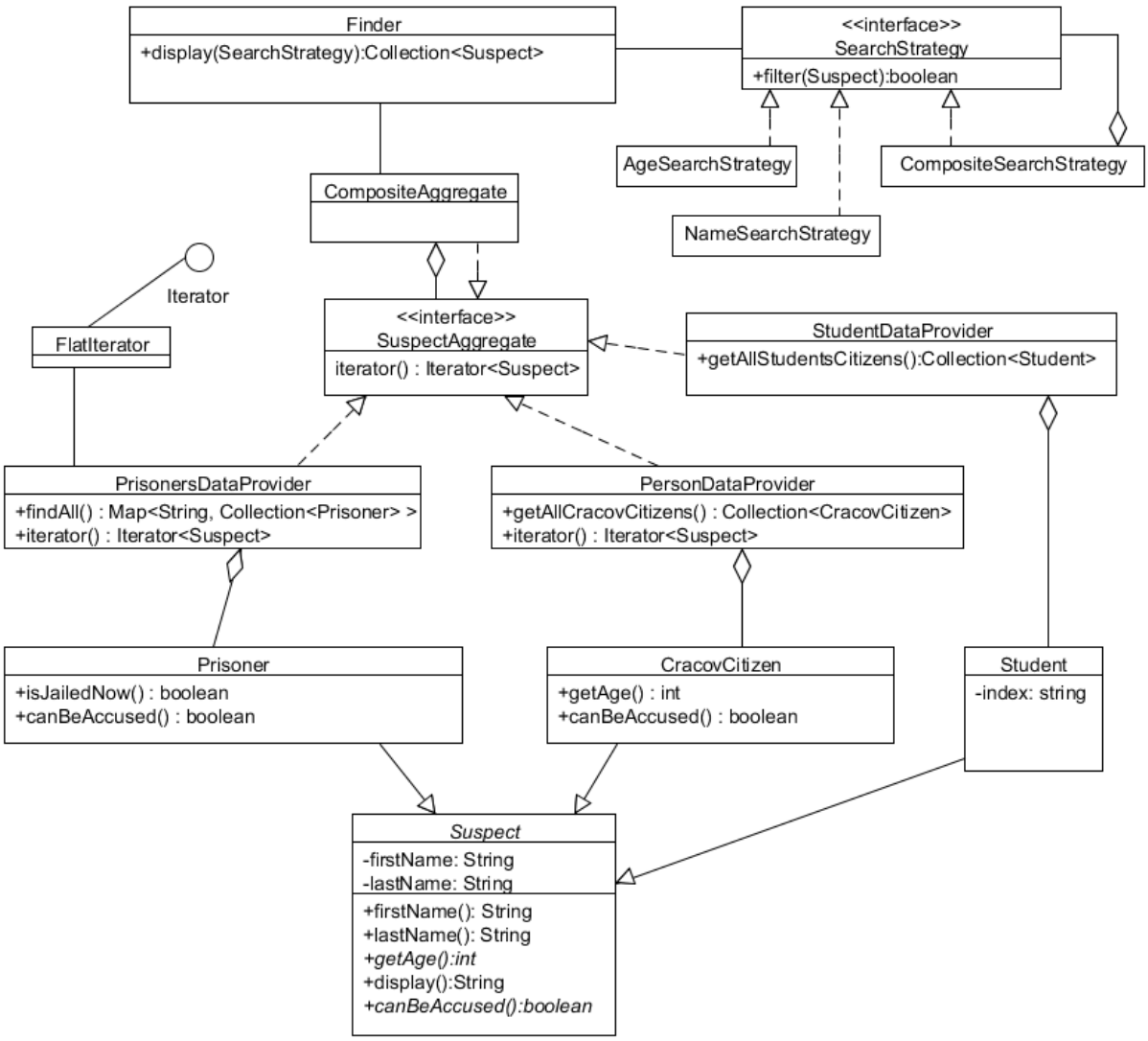
iteratora. Tym samym ściągamy z findera kolejną odpowiedzialność i jest tam coraz mniej kodu.
W jaki sposób przekazać te dane? Iteratorem, który jednocześnie agreguje wszystkie instancje z całego systemu, wykorzystując wzorzec `Composite`. Propozycję zastosowania tego wzorca przedstawia Diagram 5.



Rysunek 5: Propozycja wykorzystania kompozycji.

4.6 Krok 6

Ostatnim krokiem jest wsparcie możliwości łatwego dodawania warunków wyszukiwania. Najprostsze byłoby dodawanie kolejnych metod szukających do findera, ale jest to rozwiązania bardzo mało rozszerzalne. Lepszym rozwiązaniem będzie zastosowanie strategii szukających (zwanych też predykatami). Mają bardzo prostą implementację i, co więcej, można je łączyć ze sobą z wykorzystaniem wzorca Composite, by otrzymać złożone kryteria wyszukiwania. Wprowadzając ten wzorzec proszę dodać wyszukiwanie po wieku. Propozycję końcowej struktury aplikacji prezentuje Diagram 6.



Rysunek 6: Propozycja zastosowania strategii wyszukiwania.