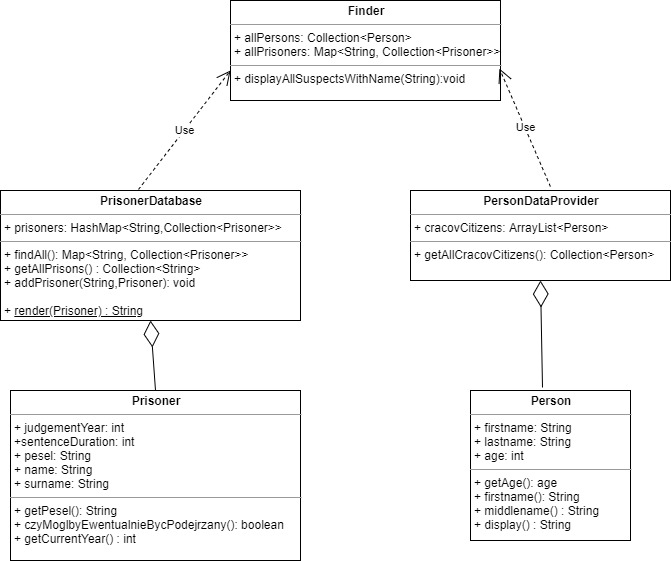
Paweł Kiełbasa, Wojciech Kosztyła

**Raport PO – Refaktoryzacja**

1. Narysowanie diagramu UML na podstawie dostarczonego kodu



Ponieważ klasy PrisonerDatabase i PersonDataProvider nie zawierają obiektów klasy Finder, realzja między mimi nie jest asocjacją , a zależnością.

1. Poprawa podstawowych błędów w kodzie (takich jak publiczne pola zamiast metod dostępowych, statyczne metody w złych miejscach, zbyt długie i enigmatyczne nazwy metod)
   1. Klasa Prisoner:
      1. Zmiana pól  i  na prywatne oraz zmiana ich nazw na „firstName” i „lastName” (w naszym przypadku bardziej czytelne).
      2. Dodanie metod dostępowych „getFirstName()”, „getLastName()” oraz „tost ring()” (zwracającą imię i nazwisko oddzielone spacją).
      3. Zmiana nazwy metody  na „isJailedNow()”, które lepiej ukazuje, co tak naprawdę wykonuje metoda.

public class Prisoner {  
 private final int judgementYear;  
  
 private final int senteceDuration;  
  
 private final String pesel;  
  
 private final String firstName;  
  
 private final String lastName;  
  
 public Prisoner(String firstName, String lastName, String pesel, int judgementYear, int sentenceDuration) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.pesel = pesel;  
 this.judgementYear = judgementYear;  
 this.senteceDuration = sentenceDuration;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public String getPesel() {  
 return pesel;  
 }  
  
 public boolean isJailedNow() {  
 return judgementYear + senteceDuration >= getCurrentYear();  
 }  
  
 public int getCurrentYear() {  
 return Calendar.*getInstance*().get(Calendar.*YEAR*);  
 }  
  
 @Override  
 public String toString() {  
 return firstName + " "+lastName;  
 }  
}

* 1. Klasa PrisonersDatabase:
     1. Zmiana nazwy metody  na „getPrisoners()”, które bardziej wyraźnie wskazuje na to, co faktycznie robi ta metoda.
     2. Usunięcie statycznej metody .

public class PrisonersDatabase {  
  
 private final Map<String, Collection<Prisoner>> prisoners = new HashMap<String, Collection<Prisoner>>();  
  
 public PrisonersDatabase() {  
 addPrisoner("Wiezienie krakowskie", new Prisoner("Jan", "Kowalski", "87080452357", 2005, 7));  
 addPrisoner("Wiezienie krakowskie", new Prisoner("Anita", "Wiercipieta", "84080452357", 2009, 3));  
 addPrisoner("Wiezienie krakowskie", new Prisoner("Janusz", "Zlowieszczy", "92080445657", 2001, 10));  
 addPrisoner("Wiezienie przedmiejskie", new Prisoner("Janusz", "Zamkniety", "802104543357", 2010, 5));  
 addPrisoner("Wiezienie przedmiejskie", new Prisoner("Adam", "Future", "880216043357", 2020, 5));  
 addPrisoner("Wiezienie przedmiejskie", new Prisoner("Zbigniew", "Nienajedzony", "90051452335", 2011, 1));  
 addPrisoner("Wiezienie centralne", new Prisoner("Jan", "Przedziwny", "91103145223", 2009, 4));  
 addPrisoner("Wiezienie centralne", new Prisoner("Janusz", "Podejrzany", "85121212456", 2012, 1));  
 }  
  
 public Map<String, Collection<Prisoner>> getPrisoners() {  
 return prisoners;  
 }  
  
 public Collection<String> getAllPrisons() {  
 return prisoners.keySet();  
 }  
  
 private void addPrisoner(String category, Prisoner prisoner) {  
 if (!prisoners.containsKey(category))  
 prisoners.put(category, new ArrayList<Prisoner>());  
 prisoners.get(category).add(prisoner);  
 }  
  
}

* 1. Klasa Person:
     1. Zmiana nazw metod ,  na „getFirstName()” i „getLastName()”, które po pierwsze: wskazują że te metody są „getterami”, a po drugie: klasa Person nie ma atrybutu „middlename”, tylko „lastName”.
     2. Zmiana nazwy metody  na „toString()”.

public class Person {  
 private String firstName;  
  
 private String lastName;  
  
 private int age;  
  
 public Person(String firstname, String lastname, int age) {  
 this.age = age;  
 this.firstName = firstname;  
 this.lastName = lastname;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public String toString() {  
 return firstName + " " + lastName;  
 }  
}

* 1. Klasa PersonDataProvider:
     1. Zmiana nazwy klasy na PersonDatabase (aby odzwierciedlała podobieństwo choćby do PrisonersDatabase).
     2. Zmiana nazwy „kolekcji ludzi” z  na .
     3. Zmiana nazwy metody  na .

public class PersonDatabase {  
  
 private final Collection<Person> cracovPeople = new ArrayList<Person>();  
  
 public PersonDatabase() {  
 cracovPersons.add(new Person("Jan", "Kowalski", 30));  
 cracovPersons.add(new Person("Janusz", "Krakowski", 30));  
 cracovPersons.add(new Person("Janusz", "Mlodociany", 10));  
 cracovPersons.add(new Person("Kasia", "Kosinska", 19));  
 cracovPersons.add(new Person("Piotr", "Zgredek", 29));  
 cracovPersons.add(new Person("Tomek", "Gimbus", 14));  
 cracovPersons.add(new Person("Janusz", "Gimbus", 15));  
 cracovPersons.add(new Person("Alicja", "Zaczarowana", 22));  
 cracovPersons.add(new Person("Janusz", "Programista", 77));  
 cracovPersons.add(new Person("Pawel", "Pawlowicz", 32));  
 cracovPersons.add(new Person("Krzysztof", "Mendel", 30));  
 }  
  
 public Collection<Person> getAllCracovPeople() {  
 return cracovPersons;  
 }  
}

* 1. Klasa Finder:
     1. Zmiana nazwy  na .
     2. Zmienił się dostęp do nazwy Prisoner’a, więc zmiana  na „prisoner.getFirstName().equals(name)”.
     3. Zmiana przy wypisywaniu podejrzanych więźniów: z  na 

public class Finder {  
 private final Collection<Person> allPeople;  
  
 private final Map<String, Collection<Prisoner>> allPrisoners;  
  
 public Finder(Collection<Person> allPeople, Map<String, Collection<Prisoner>> allPrisoners) {  
 this.allPeople = allPeople;  
 this.allPrisoners = allPrisoners;  
 }  
  
 public Finder(PersonDatabase personDataProvider, PrisonersDatabase prisonersDatabase) {  
 this(personDataProvider.getAllCracovPersons(), prisonersDatabase.getPrisoners());  
 }  
  
 public void displayAllSuspectsWithName(String name) {  
 ArrayList<Prisoner> suspectedPrisoners = new ArrayList<Prisoner>();  
 ArrayList<Person> suspectedPersons = new ArrayList<Person>();  
  
 for (Collection<Prisoner> prisonerCollection : allPrisoners.values()) {  
 for (Prisoner prisoner : prisonerCollection) {  
 if (!prisoner.isJailedNow() && prisoner.getFirstName().equals(name)) {  
 suspectedPrisoners.add(prisoner);  
 }  
 if (suspectedPrisoners.size() >= 10) {  
 break;  
 }  
 }  
 if (suspectedPrisoners.size() >= 10) {  
 break;  
 }  
 }  
  
 if (suspectedPrisoners.size() < 10) {  
 for (Person person : allPeople) {  
 if (person.getAge() > 18 && person.getFirstName().equals(name)) {  
 suspectedPersons.add(person);  
 }  
 if (suspectedPrisoners.size() + suspectedPersons.size() >= 10) {  
 break;  
 }  
 }  
 }  
  
 int t = suspectedPrisoners.size() + suspectedPersons.size();  
 System.*out*.println("Znalazlem " + t + " pasujacych podejrzanych!");  
  
 for (Prisoner n : suspectedPrisoners) {  
 System.*out*.println(n.toString());  
 }  
  
 for (Person p : suspectedPersons) {  
 System.*out*.println(p.toString());  
 }  
 }  
}

1. Propozycja generalizacji klasy Person i Prisoner
   1. Dodano klasę abstrakcyjną Suspect, która generalizuje klasy Person i Prisoner. Takie uogólnienie powinno być jak najbardziej poprawne, gdyż celem aplikacji jest pomoc służbom w znalezieniu podejrzanych, a zarówno zwykli mieszkańcy jak i więźniowie mają dużo cech wspólnych.
   2. Wybór klasy abstrakcyjnej nad interfejsem spowodowany był tym, że niektóre metody się powtarzają i nie ma sensu implementować ich zarówno w Person jak i Prisoner (to byłoby sprzeczne z zasadą DRY).
   3. Elementy, które powtarzają się w tych dwóch klasach to „firstName”, „lastName” i metody (gettery) z nimi związane. Dodano też metodę „canBeSuspected()”, która w zależności od klasy dziedziczącej ma różne implementacje.
   4. Klasa Suspect:
      1. public abstract class Suspect {  
          protected String firstName;  
          protected String lastName;  
           
          public String getFirstName() {  
          return firstName;  
          }  
           
          public String getLastName() {  
          return lastName;  
          }  
           
          @Override  
          public String toString() {  
          return firstName+" "+lastName;  
          }  
           
          public abstract boolean canBeSuspected();  
         }
   5. Klasa Person:
      1. public class Person extends Suspect {  
          private int age;  
           
          public Person(String firstname, String lastname, int age) {  
          this.age = age;  
          this.firstName = firstname;  
          this.lastName = lastname;  
          }  
           
          public int getAge() {  
          return age;  
          }  
           
          @Override  
          public boolean canBeSuspected() {  
          return age>18;  
          }  
         }
   6. Klasa Prisoner:
      1. public class Prisoner extends Suspect {  
          private final int judgementYear;  
           
          private final int senteceDuration;  
           
          private final String pesel;  
           
          public Prisoner(String firstName, String lastName, String pesel, int judgementYear, int sentenceDuration) {  
          this.firstName = firstName;  
          this.lastName = lastName;  
          this.pesel = pesel;  
          this.judgementYear = judgementYear;  
          this.senteceDuration = sentenceDuration;  
          }  
           
          public String getPesel() {  
          return pesel;  
          }  
           
          public boolean canBeSuspected() {  
          return judgementYear + senteceDuration >= getCurrentYear();  
          }  
           
          public int getCurrentYear() {  
          return Calendar.*getInstance*().get(Calendar.*YEAR*);  
          }  
           
         }
   7. Klasa Finder:
      1. Zmodyfikowano klasę Finder, aby używała zaktualizowane metody, lecz zapomnieliśmy skopiować kod (teraz jest już nadpisany).
2. Dodanie iteratora
   1. Z racji potrzeby generalizacji klas dostarczających dane do systemu utworzono interfejs „SuspectAggregate”, który zawiera iterator do „przechodzenia” po „suspectach”, jak i metodę „generateData()”, która będzie wprowadzała dane do poszczególnych struktur danych.
      1. public interface SuspectAggregate {  
          Iterator<Suspect> iterator();  
          void generateData();  
         }
   2. Zdefiniowano nowy iterator „SuspectIterator”
      1. Będzie ona uogólniała dwa różne iteratory z PrisonersDatabase i PersonDatabase.
      2. public class SuspectIterator implements Iterator<Suspect> {  
          Suspect suspect;  
          Iterator<? extends Suspect> iterator;  
           
          public SuspectIterator(Iterator<? extends Suspect> iterator){  
          this.iterator=iterator;  
          }  
           
          @Override  
          public boolean hasNext() {  
          return iterator.hasNext();  
          }  
           
          @Override  
          public Suspect next() {  
          suspect=iterator.next();  
          if(suspect != null) {  
          return suspect;  
          }  
          throw new NoSuchElementException("No such element");  
          }  
         }
   3. Zmodyfikowano PersonDatabase tak, aby implementowała SuspectAggregate.
      1. public class PersonDatabase implements SuspectAggregate {  
           
          private final Collection<Person> cracovPeople = new ArrayList<Person>();  
           
          public PersonDatabase() {}  
           
          public void generateData() {  
          cracovPeople.add(new Person("Krzysztof", "Mendel", 30));  
          cracovPeople.add(new Person("Pawel", "Pawlowicz", 32));  
          cracovPeople.add(new Person("Janusz", "Programista", 77));  
          cracovPeople.add(new Person("Alicja", "Zaczarowana", 22));  
          cracovPeople.add(new Person("Janusz", "Gimbus", 15));  
          cracovPeople.add(new Person("Tomek", "Gimbus", 14));  
          cracovPeople.add(new Person("Piotr", "Zgredek", 29));  
          cracovPeople.add(new Person("Kasia", "Kosinska", 19));  
          cracovPeople.add(new Person("Janusz", "Mlodociany", 10));  
          cracovPeople.add(new Person("Janusz", "Krakowski", 30));  
          cracovPeople.add(new Person("Jan", "Kowalski", 30));  
          }  
           
          public Collection<Person> getAllCracovPersons() {  
          return cracovPeople;  
          }  
           
          @Override  
          public Iterator<Suspect> iterator(){  
          return new SuspectIterator(cracovPeople.iterator());  
          }  
         }
   4. Zmodyfikowano PrisonersDatabase tak, aby implementowała SuspectAggregate, a także zdefiniowano dla niego iterator.
      1. public class PrisonersDatabase implements SuspectAggregate {  
           
          private final Map<String, Collection<Prisoner>> prisoners = new HashMap<String, Collection<Prisoner>>();  
           
          public PrisonersDatabase() { }  
           
          public void generateData() {  
          addPrisoner("Wiezienie krakowskie", new Prisoner("Jan", "Kowalski", "87080452357", 2005, 7));  
          addPrisoner("Wiezienie krakowskie", new Prisoner("Anita", "Wiercipieta", "84080452357", 2009, 3));  
          addPrisoner("Wiezienie krakowskie", new Prisoner("Janusz", "Zlowieszczy", "92080445657", 2001, 10));  
          addPrisoner("Wiezienie przedmiejskie", new Prisoner("Janusz", "Zamkniety", "802104543357", 2010, 5));  
          addPrisoner("Wiezienie przedmiejskie", new Prisoner("Adam", "Future", "880216043357", 2020, 5));  
          addPrisoner("Wiezienie przedmiejskie", new Prisoner("Zbigniew", "Nienajedzony", "90051452335", 2011, 1));  
          addPrisoner("Wiezienie centralne", new Prisoner("Jan", "Przedziwny", "91103145223", 2009, 4));  
          addPrisoner("Wiezienie centralne", new Prisoner("Janusz", "Podejrzany", "85121212456", 2012, 1));  
          }  
           
          public Map<String, Collection<Prisoner>> getPrisoners() {  
          return prisoners;  
          }  
           
          public Collection<String> getAllPrisons() {  
          return prisoners.keySet();  
          }  
           
          private void addPrisoner(String category, Prisoner prisoner) {  
          if (!prisoners.containsKey(category))  
          prisoners.put(category, new ArrayList<Prisoner>());  
          prisoners.get(category).add(prisoner);  
          }  
           
          @Override  
          public Iterator<Suspect> iterator() {  
          return new SuspectIterator(prisoners  
          .values()  
          .stream()  
          .flatMap(Collection::stream)  
          .collect(Collectors.*toList*())  
          .iterator());  
          }  
         }
   5. Zmodyfikowano klasę Finder, aby korzystała z nowo wprowadzonych iteratorów.
      1. public class Finder {  
          private final SuspectAggregate allPeople;  
          private final SuspectAggregate allPrisoners;  
           
          public Finder(PersonDatabase personDatabase, PrisonersDatabase prisonersDatabase) {  
          this.allPeople = personDatabase;  
          this.allPrisoners = prisonersDatabase;  
          }  
           
          public void displayAllSuspectsWithName(String name) {  
          ArrayList<Suspect> suspectedPeople = new ArrayList<Suspect>();  
          Iterator<? extends Suspect> prisonersIterator = allPrisoners.iterator();  
          Iterator<? extends Suspect> personsIterator = allPeople.iterator();  
           
          Suspect tempSuspect = null;  
          while(prisonersIterator.hasNext()){  
          tempSuspect = prisonersIterator.next();  
          if(tempSuspect.getFirstName().equals(name)&& tempSuspect.canBeSuspected()){  
          suspectedPeople.add(tempSuspect);  
          if(suspectedPeople.size() >=10) break;  
          }  
          }  
           
          if (suspectedPeople.size() < 10) {  
          while (personsIterator.hasNext()){  
          tempSuspect = personsIterator.next();  
          if (tempSuspect.canBeSuspected() && tempSuspect.getFirstName().equals(name)) {  
          suspectedPeople.add(tempSuspect);  
          }  
          if (suspectedPeople.size() >= 10) {  
          break;  
          }  
          }  
          }  
           
          System.*out*.println("Znaleziono " + suspectedPeople.size() + " pasujacych podejrzanych!");  
          for (Suspect suspect:suspectedPeople) {  
          System.*out*.println(suspect.toString());  
          }  
          }  
         }
3. Dodanie klasy pośredniej między agregatami, a Finder’em: CompositeAggregate
   1. Klasa ta ściąga z Finder’a odpowiedzialność zbierania struktur danych i podaje mu wszystko jako jedna duża lista.
      1. public class CompositeAggregate implements SuspectAggregate {  
          private final List<SuspectAggregate> databases;  
           
          public CompositeAggregate(List<SuspectAggregate> databases){  
          this.databases=databases;  
          }  
          @Override  
          public Iterator<Suspect> iterator() {  
          Collection<Suspect> suspects =new ArrayList<>();  
          databases.forEach(data ->{  
           
          Iterator<Suspect> iterator =data.iterator();  
          while ((iterator.hasNext())){  
          suspects.add(iterator.next());  
          }  
           
          });  
          return suspects.iterator();  
          }  
         }
   2. I oczywiście modyfikowana jest też klasa Finder
      1. public class Finder {  
          private final CompositeAggregate compositeAggregate;  
           
           
          public Finder(CompositeAggregate compositeAggregate) {  
          this.compositeAggregate =compositeAggregate;  
          }  
           
          public void displayAllSuspectsWithName(String name) {  
          ArrayList<Suspect> suspectPeople = new ArrayList<Suspect>();  
          Iterator<? extends Suspect> suspectIterator = compositeAggregate.iterator();  
           
          Suspect tempSuspect = null;  
          while(suspectIterator.hasNext()){  
          tempSuspect = suspectIterator.next();  
          if(tempSuspect.getFirstName().equals(name)&& tempSuspect.canBeSuspected()){  
          suspectPeople.add(tempSuspect);  
          if(suspectPeople.size() >=10) break;  
          }  
          }  
          System.*out*.println("Znalaziono " + suspectPeople.size() + " pasujacych podejrzanych!");  
          for (Suspect suspect:suspectPeople) {  
          System.*out*.println(suspect.toString());  
          }  
          }  
         }
4. Zmiana sposobu wyszukiwania oraz dodanie nowej klasy Student
   1. Dodano interfejs SearchStrategy z metodą filter pozwalającą na sprawdziedzi czy dany podejrzany spełnia wybrane warunki

public interface SearchStrategy {  
 boolean filter(Suspect suspect);  
}

* 1. Dodano klasę NameSearchStrategy która implementuje interjejs SearchStrategy, w której szukamy podejrzanych względem nazwy

public class NameSearchStrategy implements SearchStrategy {  
 private String name;  
  
 public NameSearchStrategy(String name){  
 this.name = name;  
 }  
  
 @Override  
 public boolean filter(Suspect suspect) {  
 return suspect.getFirstName().equals(name);  
 }  
}

* 1. Dodano klasę AgeStrategySearch z wyszukiwaniem względem wieku podejrzanego

public class AgeSearchStrategy implements SearchStrategy {  
 private int age;  
  
 public AgeSearchStrategy(int age){  
 this.age=age;  
 }  
  
 @Override  
 public boolean filter(Suspect suspect) {  
 if(suspect instanceof Person) return this.age == ((Person)suspect).getAge();  
 return false;  
 }  
}

* 1. Dodano klasę CompositeStrategySearch w której dodano możliwość tworzenia list parametrów wyszukiwania z klas NameStrategySearch i AgeStrategySearch

public class CompositeSearchStrategy implements SearchStrategy {  
 private final List<SearchStrategy> filters;  
  
 public CompositeSearchStrategy(List<SearchStrategy> filters){  
 this.filters=filters;  
 }  
  
 @Override  
 public boolean filter(Suspect suspect) {  
 return filters.stream().allMatch(s-> s.filter(suspect));  
 }  
}

* 1. W związku z powyższymi zmianami wprowadzono zmiany w klasie SuspectIterator, która teraz przechowuje również pole searchStrategy. W metodzie hasNext w związku z tym zwracamy prawdę tylko wtedy, gdy podejrzany spełnia określone warunki filtrowania i może być sądzony

public class SuspectIterator implements Iterator<Suspect> {  
 Suspect suspect;  
 Iterator<? extends Suspect> iterator;  
 SearchStrategy searchStrategy;  
  
 public SuspectIterator(Iterator<? extends Suspect> iterator, SearchStrategy searchStrategy){  
 this.iterator=iterator;  
 this.searchStrategy =searchStrategy;  
 }  
  
 @Override  
 public boolean hasNext() {  
 while (iterator.hasNext()){  
 Suspect tempSuspect = iterator.next();  
 if(searchStrategy.filter(tempSuspect)&& tempSuspect.canBeSuspected()){  
 suspect = tempSuspect;  
 return true;  
 }  
 }  
 return false;  
 }  
  
 @Override  
 public Suspect next() {  
 suspect=iterator.next();  
 if(suspect != null) {  
 return suspect;  
 }  
 throw new NoSuchElementException("No such element");  
 }  
}

* 1. Dodano w interfejsie SuspectAggregate w metodzie iterator dodano jako parametr element klasy SearchStrategy. Wymaga to również poprawy wszystkich klas implementujących dany interfejs

public interface SuspectAggregate {  
 Iterator<Suspect> iterator(SearchStrategy searchStrategy);  
}

Klasa PrisonersDatabase

public class PrisonersDatabase implements SuspectAggregate {  
  
…

@Override  
 public Iterator<Suspect> iterator(SearchStrategy searchStrategy) {  
 return new SuspectIterator(prisoners  
 .values()  
 .stream()  
 .flatMap(Collection::stream)  
 .collect(Collectors.*toList*())  
 .iterator(),searchStrategy);  
 }  
}

Klasa CompositeAggregate

public class CompositeAggregate implements SuspectAggregate {  
 private final List<SuspectAggregate> databases;  
  
 public CompositeAggregate(List<SuspectAggregate> databases){  
 this.databases=databases;  
 }  
 @Override  
 public Iterator<Suspect> iterator(SearchStrategy searchStrategy) {  
 Collection<Suspect> suspects =new ArrayList<>();  
 databases.forEach(data ->{  
 Iterator<Suspect> iterator =data.iterator(searchStrategy);  
 while ((iterator.hasNext())){  
 suspects.add(iterator.next());  
 }  
 });  
 return suspects.iterator();  
 }  
  
  
}

Klasa PersonDatabase

public class PersonDatabase implements SuspectAggregate {  
  
…  
  
 @Override  
 public Iterator<Suspect> iterator(SearchStrategy searchStrategy){  
 return new SuspectIterator(cracovPeople.iterator(),searchStrategy);  
 }  
}

* 1. W klasie Finder wprowadzono zmiany służące do obsługi dodanych filtrów w powyższych podpunktach

public class Finder {  
 private final CompositeAggregate compositeAggregate;  
  
 public Finder(CompositeAggregate compositeAggregate) {  
 this.compositeAggregate = compositeAggregate;  
 }  
  
 public void display(SearchStrategy searchStrategy) {  
 ArrayList<Suspect> suspectPeople = new ArrayList<Suspect>();  
 Iterator<Suspect> suspectIterator = compositeAggregate.iterator(searchStrategy);  
  
 while (suspectIterator.hasNext()) {  
 suspectPeople.add(suspectIterator.next());  
 }  
  
 System.*out*.println("Znalazlem " + suspectPeople.size() + " pasujacych podejrzanych!");  
  
 for (Suspect suspect: suspectPeople) {  
 System.*out*.println(suspect.toString());  
 }  
 }  
}

* 1. Dodanie klasy Student i StudentDatabase zgodnie z wymogami w instrukcji

public class Student extends Suspect {  
 private String index;  
  
 public Student(String firstName, String lastName, String index){  
 this.firstName=firstName;  
 this.lastName=lastName;  
 this.index=index;  
 }  
  
 public String getIndex() {  
 return index;  
 }  
  
 @Override  
 public boolean canBeSuspected() {  
 return true;  
 }  
}

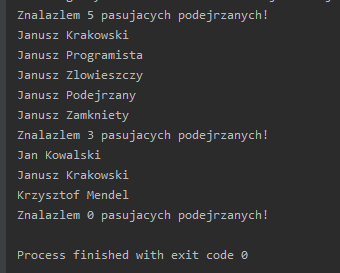
public class StudentDatabase implements SuspectAggregate {  
 private final Collection<Student> students = new ArrayList<Student>();  
  
 public StudentDatabase() { }  
  
 @Override  
 public Iterator<Suspect> iterator(SearchStrategy searchStrategy) {  
 return new SuspectIterator(students.iterator(), searchStrategy);  
 }  
  
 public void generateData() {  
 addStudent("Bro", "Fida", "832782");  
 addStudent("Wadim", "Kosman", "332474");  
 addStudent("Remigiusz", "Pylka", "845446");  
 addStudent("Konrad", "Kasza", "234523");  
}

public Collection<Student> getStudents() {  
 return students;  
 }  
  
 public void addStudent(String firstName, String lastName, String index) {  
 students.add(new Student(firstName, lastName, index));  
 }  
}

* 1. Zmiana w klasie Application pozwalająca na wywołanie programu

public class Application {  
  
 public static void main(String[] args) {  
 PersonDatabase personDatabase = new PersonDatabase();  
 personDatabase.generateData();  
 PrisonersDatabase prisonerDatabase = new PrisonersDatabase();  
 prisonerDatabase.generateInitialData();  
 StudentDatabase studentDatabase = new StudentDatabase();  
 studentDatabase.generateData();  
  
 List<SuspectAggregate> databases = new ArrayList<>();  
 databases.add(personDatabase);  
 databases.add(prisonerDatabase);  
 databases.add(studentDatabase);  
  
 Finder suspects = new Finder(new CompositeAggregate(databases));  
  
 suspects.display(new NameSearchStrategy("Janusz"));  
 suspects.display(new AgeSearchStrategy(30));  
 List<SearchStrategy> strategies = new ArrayList<>();  
 strategies.add(new NameSearchStrategy("Tomek"));  
 strategies.add(new AgeSearchStrategy(14));  
 suspects.display(new CompositeSearchStrategy(strategies));  
 }  
}

Wynik działania programu



W celu sprawdzenia poprawności działania programu zostały przeprowadzone testy. Testy uzyskały następujące rezultaty

