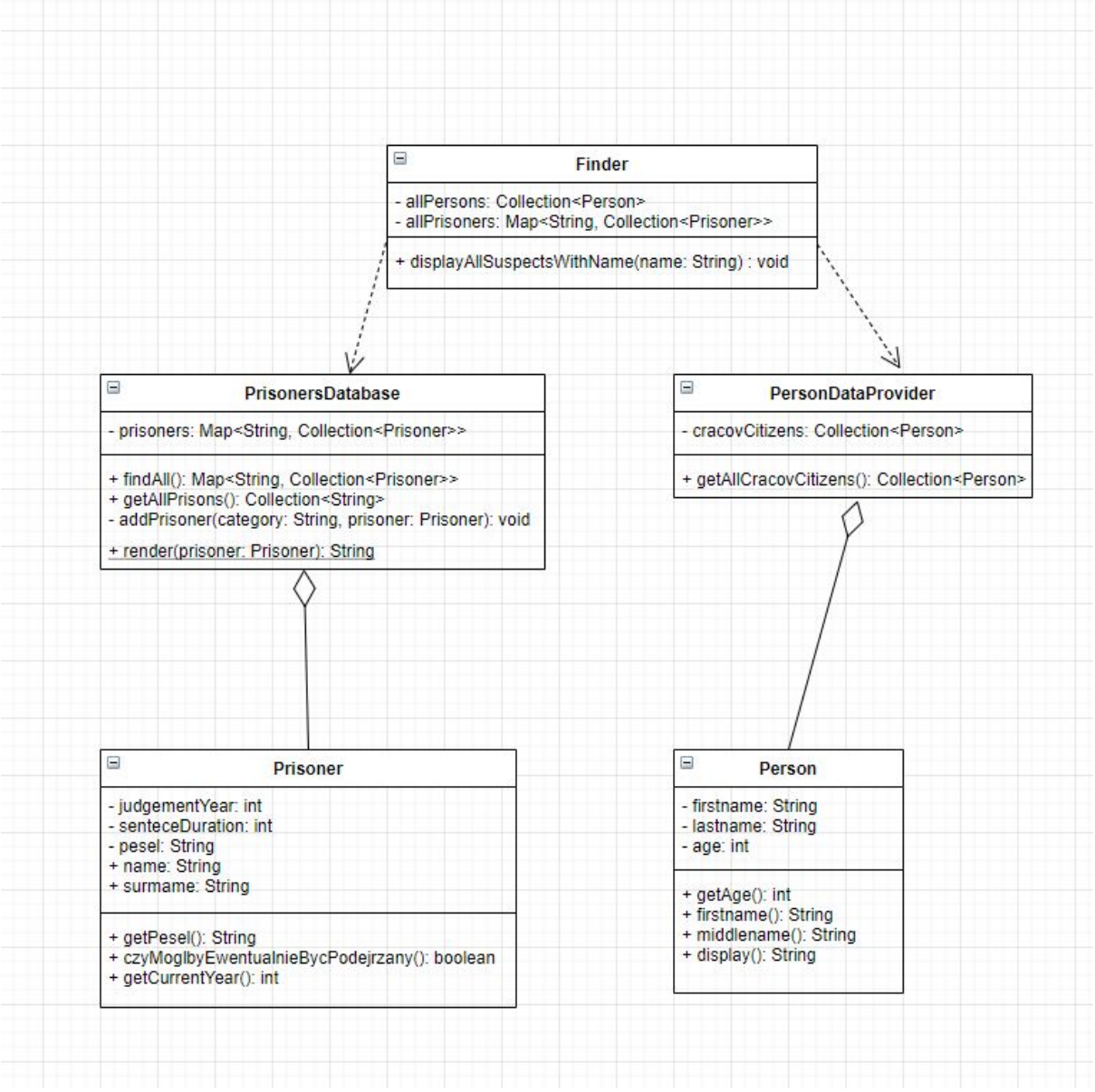
Laboratorium 4
Adrian Wójciak, Lena Obertyńska

Krok 1

Schemat na podstawie analizy kodu:



Różnice w stosunku do schematu z instrukcji:

- wylistowano wszystkie własności/metody, nie tylko te publiczne
- relacje między klasą Finder a klasami PrisonersDatabase i PersonDataProvider zostały zmienione z dwukierunkowych asocjacji na jednokierunkowe zależności - Finder używa obydwu, natomiast ani on nie posiada atrubutu klasy PrisonersDatabase/PersonDataProvider, ani żadna z nich atrybutu klasy Finder

Propozycje zmian po analizie kodu:

- ujednolicenie i uproszczenie nazewnictwa metod dostępowych - użycie form get...
- zapis wszystkich nazw własności/metod w języku angielskim
- rezygnacja z pól publicznych na rzecz metod dostępowych
- uspójnienie nazewnictwa klas/metod/własności

Krok 2

a) klasa Person

Nazwa klasy została zmieniona na CracovCitizen, co lepiej oddaje funkcjonalność; nazwy własności firstname i lastname zostały zmienione na name i surname celem uspójnienia nazewnictwa analogicznych własności w klasach, ponadto nazwy ich metod dostępowych zawierają teraz przedrostek get.

```
public class CracovCitizen {
   private String name;

   private String surname;

   private int age;

   public CracovCitizen(String name, String surname, int age) {
       this.age = age;
       this.name = name;
       this.surname = surname;
   }

   public int getAge() {
       return age;
   }

   public String getName() {
       return name;
   }

   public String getSurname() {
       return surname;
   }

   public String display() {
       return name + " " + surname;
   }
}
```

b) klasa Prisoner

Pola name i surname stały się prywatne, ponadto doczekały się metod dostępowych; metoda czyMoglbyEwentualnieBycPodejrzany dostała nową nazwę w języku angielskim -

isJailedNow - która też lepiej oddaje funkcjonalność; została utworzona metoda display, do której przeniesiono funkcjonalność metody render z klasy PrisonersDatabase.

```
public class Prisoner {
   private final int judgementYear;

   private final int sentenceDuration;

   private final String pesel;

   private final String name;

   private final String surname;

   public Prisoner(String name, String surname, String pesel, int
judgementYear, int sentenceDuration) {
       this.name = name;
       this.surname = surname;
       this.pesel = pesel;
       this.judgementYear = judgementYear;
       this.sentenceDuration = sentenceDuration;
   }

   public String getName() {
       return name;
   }

   public String getSurname() {
       return surname;
   }

   public String getPesel() {
       return pesel;
   }

   public boolean isJailedNow() {
       return judgementYear + sentenceDuration >= getCurrentYear();
   }

   public int getCurrentYear() {
       return Calendar.getInstance().get(Calendar.YEAR);
   }

   public String display() {
       return name + " " + surname;
   }
```

```
}
```

c) klasa PersonDataProvider

Klasa zmieniła nazwę na CracovCitizensDatabase celem uspójnienia nazewnictwa analogicznych klas. Nazwa metody dostępowej dla pola cracowCitizens uproszczona została dzięki usunięciu z niej słowa All.

```
public class CracovCitizensDatabase {

   private final Collection<CracovCitizen> cracovCitizens = new
ArrayList<CracovCitizen>();

   public CracovCitizensDatabase() {
       cracovCitizens.add(new CracovCitizen("Jan", "Kowalski", 30));
       cracovCitizens.add(new CracovCitizen("Janusz", "Krakowski", 30));
       cracovCitizens.add(new CracovCitizen("Janusz", "Mlodociany", 10));
       cracovCitizens.add(new CracovCitizen("Kasia", "Kosinska", 19));
       cracovCitizens.add(new CracovCitizen("Piotr", "Zgredek", 29));
       cracovCitizens.add(new CracovCitizen("Tomek", "Gimbus", 14));
       cracovCitizens.add(new CracovCitizen("Janusz", "Gimbus", 15));
       cracovCitizens.add(new CracovCitizen("Alicja", "Zaczarowana", 22));
       cracovCitizens.add(new CracovCitizen("Janusz", "Programista", 77));
       cracovCitizens.add(new CracovCitizen("Pawel", "Pawlowicz", 32));
       cracovCitizens.add(new CracovCitizen("Krzysztof", "Mendel", 30));
   }

   public Collection<CracovCitizen> getCracovCitizens() {
       return cracovCitizens;
   }
}
```

d) klasa PrisonersDatabase

Nazwa metody dostępowej dla pola prisoners została wzorem innych zmieniona na getPrisoners; z nazwy metody getAllPrisons pozbyto się słowa All; funkcjonalność metody render przeniesiona zostala do klasy Prisoner do metody display.

```
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> getPrisons() {
        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);
    }
}
```

e) klasa Finder

Z nazw pól i metod pozbyto się słowa All celem uproszczenia nazewnictwa.

```
public class Finder {
    private final Collection<CracovCitizen> cracovCitizens;

    private final Map<String, Collection<Prisoner>> prisoners;

    public Finder(Collection<CracovCitizen> cracovCitizens, Map<String,
Collection<Prisoner>> prisoners) {
        this.cracovCitizens = cracovCitizens;
        this.prisoners = prisoners;
    }

    public Finder(CracovCitizensDatabase cracovCitizensDatabase,
PrisonersDatabase prisonersDatabase) {
```

```
        this(cracovCitizensDatabase.getCracovCitizens(),
prisonersDatabase.getPrisoners());
    }

    public void displaySuspectsWithName(String name) {
        ArrayList<Prisoner> suspectedPrisoners = new ArrayList<Prisoner>();
        ArrayList<CracovCitizen> suspectedCracovCitizens = new
ArrayList<CracovCitizen>();

        for (Collection<Prisoner> prisonerCollection : prisoners.values())
{
            for (Prisoner prisoner : prisonerCollection) {
                if (!prisoner.isJailedNow() &&
(prisoner.getName()).equals(name)) {
                    suspectedPrisoners.add(prisoner);
                }
                if (suspectedPrisoners.size() >= 10) {
                    break;
                }
            }
            if (suspectedPrisoners.size() >= 10) {
                break;
            }
        }

        if (suspectedPrisoners.size() < 10) {
            for (CracovCitizen cracovCitizen : cracovCitizens) {
                if (cracovCitizen.getAge() > 18 &&
(cracovCitizen.getName()).equals(name)) {
                    suspectedCracovCitizens.add(cracovCitizen);
                }
                if (suspectedPrisoners.size() +
suspectedCracovCitizens.size() >= 10) {
                    break;
                }
            }
        }

        int t = suspectedPrisoners.size() + suspectedCracovCitizens.size();
        System.out.println("Znalazlem " + t + " pasujacych podejrzanych!");

        for (Prisoner n : suspectedPrisoners) {
            System.out.println(n.display());
        }

        for (CracovCitizen p : suspectedCracovCitizens) {
```

```
            System.out.println(p.display());
        }
    }
}
```

Ponadto w każdej z klas zostały dokonane zmiany wynikające z wymienionych powyżej (np. odwoływanie się do własności przez metody dostępowe, a nie bezpośrednio, zmiany nazw wywoływanych metod, itp.)

Krok 3

Do generalizacji klas Person (teraz CracovCitizen) i Prisoner wykorzystana została klasa abstrakcyjna - pola w interfejsach mogą mieć tylko widoczność public, co nie jest dobrym rozwiązaniem, gdy posługujemy się konwencją, według której używamy prywatnych pól, których wartości pozyskujemy przy użyciu prywatnych metod.
Celem uogólnienia klasy Finder stworzyliśmy metodę isProperSuspect, która zwraca koniunkcję wyniku warunku '...getName().equals(name)" oraz:

- dla klasy Prisoner - zaprzeczenia wartości funkcji isJailedNow
- dla klasy CracovCitizen wartości warunku "cracovCitizen.getAge() > 18", sprawdzanego w analogicznej instrukcji warunkowej w klasie Finder

Klasa Suspect

```
public abstract class Suspect {
    private String name;

    private String surname;

    public Suspect(String name, String surname) {
        this.name = name;
        this.surname = surname;
    }

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }

    public String display() {
        return name + " " + surname;
    }

    public boolean isProperSuspect(String givenName) {
```

```
        return givenName.equals(name);
    }
}
```

Klasa CracovCitizen

```
public class CracovCitizen extends Suspect {
    private int age;

    public CracovCitizen(String name, String surname, int age) {
        super(name, surname);
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    @Override
    public boolean isProperSuspect(String givenName) {
        return super.isProperSuspect(givenName) && (age > 18);
    }
}
```

Klasa Prisoner

```
public class Prisoner extends Suspect {
    private final int judgementYear;

    private final int sentenceDuration;

    private final String pesel;

    public Prisoner(String name, String surname, String pesel, int
judgementYear, int sentenceDuration) {
        super(name, surname);
        this.pesel = pesel;
        this.judgementYear = judgementYear;
        this.sentenceDuration = sentenceDuration;
    }

    public String getPesel() {
        return pesel;
    }

    public boolean isJailedNow() {
```

```
        return judgementYear + sentenceDuration >= getCurrentYear();
    }

    @Override
    public boolean isProperSuspect(String givenName) {
        return super.isProperSuspect(givenName) && !isJailedNow();
    }

    public int getCurrentYear() {
        return Calendar.getInstance().get(Calendar.YEAR);
    }
}
```

Klasa Finder

```
public class Finder {
    private final Collection<CracovCitizen> cracovCitizens;

    private final Map<String, Collection<Prisoner>> prisoners;

    public Finder(Collection<CracovCitizen> cracovCitizens, Map<String,
Collection<Prisoner>> prisoners) {
        this.cracovCitizens = cracovCitizens;
        this.prisoners = prisoners;
    }

    public Finder(CracovCitizensDatabase cracovCitizensDatabase,
PrisonersDatabase prisonersDatabase) {
        this(cracovCitizensDatabase.getCracovCitizens(),
prisonersDatabase.getPrisoners());
    }

    public void displaySuspectsWithName(String name) {
        ArrayList<Prisoner> suspectedPrisoners = new ArrayList<Prisoner>();
        ArrayList<CracovCitizen> suspectedCracovCitizens = new
ArrayList<CracovCitizen>();

        for (Collection<Prisoner> prisonerCollection : prisoners.values())
{
            for (Prisoner prisoner : prisonerCollection) {
                if (prisoner.isProperSuspect(name)) {
                    suspectedPrisoners.add(prisoner);
                }
                if (suspectedPrisoners.size() >= 10) {
                    break;
```

```
                }
            }
            if (suspectedPrisoners.size() >= 10) {
                break;
            }
        }

        if (suspectedPrisoners.size() < 10) {
            for (CracovCitizen cracovCitizen : cracovCitizens) {
                if (cracovCitizen.isProperSuspect(name)) {
                    suspectedCracovCitizens.add(cracovCitizen);
                }
                if (suspectedPrisoners.size() +
suspectedCracovCitizens.size() >= 10) {
                    break;
                }
            }
        }

        int t = suspectedPrisoners.size() + suspectedCracovCitizens.size();
        System.out.println("Znalazlem " + t + " pasujacych podejrzanych!");

        for (Prisoner n : suspectedPrisoners) {
            System.out.println(n.display());
        }

        for (CracovCitizen p : suspectedCracovCitizens) {
            System.out.println(p.display());
        }
    }
}
```

Krok 4

Do rozwiązania problemu odmiennej struktury danych zwracanej przez dostawców wykorzystano Iterator. W przypadku CracovCitizensDatabase możemy wykorzystać Iterator dostarczony przez interfejs Collection. W przypadku PrisonersDatabase musimy zaimplementować swój Iterator. Stworzono klasę FlatIterator która implementuje interfejs Iterator. Również został dodany interfejs SuspectAggregate który implementują klasy PrisonersDatabase i CracovCitizensDatabase. Do klasy Finder zostały wprowadzone zmiany tak, by uwzględnić refaktoryzację opisaną wyżej.

```
package pl.edu.agh.to.lab4;
import java.util.Iterator;

public interface SuspectAggregate {
    Iterator<Suspect> iterator();
```

```
}
```

```
public class CracovCitizensDatabase implements SuspectAggregate{

   private final Collection<Suspect> cracovCitizens = new
ArrayList<Suspect>();

   public CracovCitizensDatabase() {
       cracovCitizens.add(new CracovCitizen("Jan", "Kowalski",
30));
       cracovCitizens.add(new CracovCitizen("Janusz", "Krakowski",
30));
       cracovCitizens.add(new CracovCitizen("Janusz", "Mlodociany",
10));
       cracovCitizens.add(new CracovCitizen("Kasia", "Kosinska",
19));
       cracovCitizens.add(new CracovCitizen("Piotr", "Zgredek",
29));
       cracovCitizens.add(new CracovCitizen("Tomek", "Gimbus",
14));
       cracovCitizens.add(new CracovCitizen("Janusz", "Gimbus",
15));
       cracovCitizens.add(new CracovCitizen("Alicja",
"Zaczarowana", 22));
       cracovCitizens.add(new CracovCitizen("Janusz",
"Programista", 77));
       cracovCitizens.add(new CracovCitizen("Pawel", "Pawlowicz",
32));
       cracovCitizens.add(new CracovCitizen("Krzysztof", "Mendel",
30));
   }

   public Iterator<Suspect> iterator(){
       Iterator<Suspect> iterator=cracovCitizens.iterator();
       return iterator;
   }

   public Collection<Suspect> getCracovCitizens() {
       return cracovCitizens;
   }
}
```

```
public class PrisonersDatabase implements SuspectAggregate,
Iterable<Suspect>{

   private final Map<String, Collection<Suspect>> prisoners = new
HashMap<String, Collection<Suspect>>();

   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));
   }
   @Override
   public Iterator<Suspect> iterator(){
       return new FlatIterator(this.prisoners);
   }

   public Map<String, Collection<Suspect>> getPrisoners() {
       return prisoners;
   }

   public Collection<String> getPrisons() {
       return prisoners.keySet();
   }

   private void addPrisoner(String category, Prisoner prisoner) {
       if (!prisoners.containsKey(category))
           prisoners.put(category, new ArrayList<Suspect>());
       prisoners.get(category).add(prisoner);
   }
}
```

```
public class FlatIterator implements Iterator<Suspect> {
   private Map<String, Collection<Suspect>> prisoners;
   private Iterator<String> prisonsIterator;
   private Iterator<Suspect> suspectsIterator;

   public FlatIterator(Map<String, Collection<Suspect>> prisoners){
       this.prisoners=prisoners;
       this.prisonsIterator=this.prisoners.keySet().iterator();

this.suspectsIterator=this.prisoners.get(this.prisonsIterator.next()
).iterator();
   }

   @Override
   public boolean hasNext() {
       if(this.suspectsIterator.hasNext()) return true;
       else{
           if(!this.prisonsIterator.hasNext()) return false;
           else{

this.suspectsIterator=this.prisoners.get(this.prisonsIterator.next()
).iterator();
               return true;
           }
       }
   }

   @Override
   public Suspect next()throws NoSuchElementException{
       return this.suspectsIterator.next();
   }
}


public class Finder {
   private Iterator<Suspect> cracovCitizens;
   private Iterator<Suspect> prisoners;

 public Finder(Iterator<Suspect> cracovCitizens, Iterator<Suspect>
prisoners) {
     this.cracovCitizens = cracovCitizens;
     this.prisoners = prisoners;
```

```
 }

   public Finder(CracovCitizensDatabase cracovCitizensDatabase,
PrisonersDatabase prisonersDatabase) {
       this(cracovCitizensDatabase.iterator(),
prisonersDatabase.iterator());
   }


   public void displaySuspectsWithName(String name) {
       ArrayList<Suspect> suspectedPrisoners = new
ArrayList<Suspect>();
       ArrayList<Suspect> suspectedCracovCitizens = new
ArrayList<Suspect>();

       while(suspectedPrisoners.size() < 10 &&
this.prisoners.hasNext()){
           Suspect prisoner=this.prisoners.next();
           if (prisoner.isProperSuspect(name)) {
               suspectedPrisoners.add(prisoner);
           }
       }

       while(suspectedCracovCitizens.size() < 10 &&
this.cracovCitizens.hasNext()){
           Suspect cracovCitizen=this.cracovCitizens.next();
           if (cracovCitizen.isProperSuspect(name)) {
               suspectedCracovCitizens.add(cracovCitizen);
           }
       }

       int t = suspectedPrisoners.size() +
suspectedCracovCitizens.size();
       System.out.println("Znalazlem " + t + " pasujacych
podejrzanych!");

       for (Suspect n : suspectedPrisoners) {
           System.out.println(n.display());
       }

       for (Suspect p : suspectedCracovCitizens) {
           System.out.println(p.display());
       }
   }
}
```

Krok 5
Żeby zredukować problem dodania nowego zbioru dodano klasę pośrednią pomiędzy agregatami a finderem - CompositeAggrigate. Odpowiedzialnością tej klasy jest agregacja wszystkich osób ze zbiorów danych i przekazanie ich do findera przy pomocy iteratora.

```
public class CompositeAggrigate implements Iterator<Suspect> {
   private List<Iterator<Suspect>> iterators=new ArrayList<>();
   private Iterator<Suspect> currentIterator;
   private int iteratorNumber;

   public void addData(Iterator<Suspect> suspectsIterator){
       this.iterators.add(suspectsIterator);
   }
   public void resetIterator(){
       this.iteratorNumber=0;
       this.currentIterator=this.iterators.get(iteratorNumber);
   }

   public boolean hasNext(){
       if(this.iteratorNumber<this.iterators.size()-1) {
           if (!this.currentIterator.hasNext()) {
               this.iteratorNumber++;
               this.currentIterator =
this.iterators.get(this.iteratorNumber);
           }
           return true;
       }
       return this.currentIterator.hasNext();
   }

   public Suspect next()throws NoSuchElementException{
       return this.currentIterator.next();
   }
}



public class Finder {

   CompositeAggrigate data=new CompositeAggrigate();

   public void addDataIterator(Iterator<Suspect> iterator) {
       this.data.addData(iterator);
   }
```

```
    public void displaySuspectsWithName(String name) {
        data.resetIterator();
        ArrayList<Suspect> suspected = new ArrayList<Suspect>();

        while(suspected.size() < 10 && this.data.hasNext()){
            Suspect suspect=this.data.next();
            if (suspect.isProperSuspect(name)) {
                suspected.add(suspect);
            }
        }

        int t = suspected.size();
        System.out.println("Znalazlem " + t + " pasujacych
podejrzanych!");

        for (Suspect n : suspected) {
            System.out.println(n.display());
        }
    }
}
```

Jak widać, klasa Finder zawiera coraz mniej kodu.

Krok 6

W ostatnim kroku umożliwiono łatwe dodawanie warunków wyszukiwania. Stworzony został interfejs SearchStrtegy i klasy implementujące: AgeSearchStrategy, NameSearchStrategy oraz CompositeSearchStrategy. Również dodano klasy StudentsDatabase i Student.

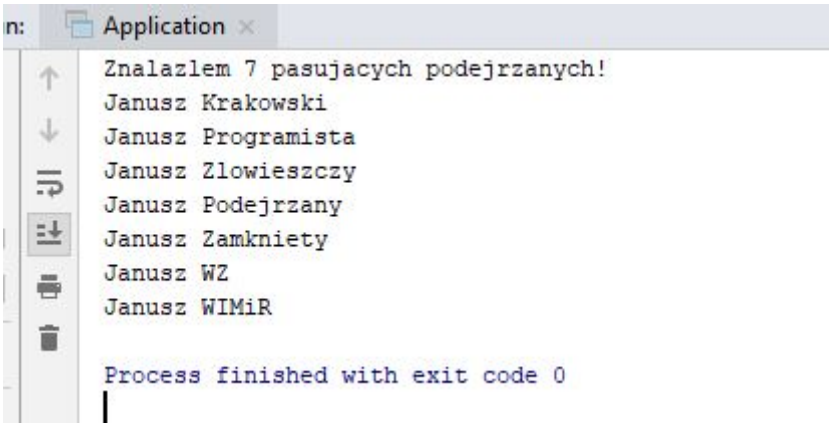
```
public class StudentsDatabase implements SuspectAggregate {
    private final Collection<Suspect> students = new ArrayList<>();


    public StudentsDatabase(){
        students.add(new Student("Jan", "WIEiT", "30"));
        students.add(new Student("Janusz", "WZ", "301"));
        students.add(new Student("Janusz", "WIMiR", "90"));
        students.add(new Student("Kasia", "WH", "19"));
    }

    public Iterator<Suspect> iterator(){
        return this.students.iterator();
    }
}
```

```
public class Student extends Suspect {
   private String index;

   public Student(String name, String surname, String index){
        super(name,surname);
        this.index=index;
   }
   @Override
   public boolean isProperSuspect(){
        return super.isProperSuspect();
   }
   @Override
   public int getAge(){return 20;}
}


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


public class NameSearchStrategy implements SearchStrategy{
   private String name;

   public NameSearchStrategy(String name){ this.name=name; }

   @Override
   public boolean filter(Suspect suspect){
        if(suspect.isProperSuspect() &&
suspect.getName().equals(this.name)) return true;
        return false;
   }
}


public class AgeSearchStrategy implements SearchStrategy{
   int age;

   public AgeSearchStrategy(int age){
        this.age=age;
   }

   @Override
   public boolean filter(Suspect suspect){
```

```
        if(suspect.isProperSuspect() && suspect.getAge()>age) return
true;
        return false;
    }
}


public class CompositeSearchStrategy implements SearchStrategy{

    private NameSearchStrategy nameSearch;
    private AgeSearchStrategy ageSearch;

    public CompositeSearchStrategy(String name){
        this.nameSearch=new NameSearchStrategy(name);
    }
    public CompositeSearchStrategy(String name, int age){
        this.nameSearch=new NameSearchStrategy(name);
        this.ageSearch=new AgeSearchStrategy(age);
    }
    public CompositeSearchStrategy(int age){
        this.ageSearch=new AgeSearchStrategy(age);
    }

    @Override
    public boolean filter(Suspect suspect){
        if(this.nameSearch!=null){
            if(!this.nameSearch.filter(suspect)) return false;
        }
        if(this.ageSearch!=null){
            if(!this.ageSearch.filter(suspect)) return false;
        }
        return true;
    }
}



public class Finder {

    CompositeAggrigate data=new CompositeAggrigate();

    public void addDataIterator(Iterator<Suspect> iterator) {
        this.data.addData(iterator);
    }
```

```
    public void filtering(SearchStrategy searchStrategy){
        data.resetIterator();
        ArrayList<Suspect> suspected = new ArrayList<Suspect>();
        while(suspected.size() < 10 && this.data.hasNext()){
            Suspect suspect=this.data.next();

            if (searchStrategy.filter(suspect)) {
                suspected.add(suspect);
            }
        }

        int t = suspected.size();
        System.out.println("Znalazlem " + t + " pasujacych
podejrzanych!");

        for (Suspect n : suspected) {
            System.out.println(n.display());
        }
    }

    public void displaySuspects(String name, int age) {
        SearchStrategy searchStrategy=new
CompositeSearchStrategy(name,age);
        filtering(searchStrategy);
    }
    public void displaySuspects(String name) {
        SearchStrategy searchStrategy=new
CompositeSearchStrategy(name);
        filtering(searchStrategy);
    }
}
```

Szukając po wieku i imieniu wśród studentów, mieszkańców Krakowa oraz więźniów znaleziono 7 podejrzanych:


n:
Application
Znalazlem 7 pasujacych podejrzanych!
Janusz Krakowski
Janusz Programista
Janusz Zlowieszczy
Janusz Podejrzany
Janusz Zamkniety
Janusz WZ
Janusz WIMiR
Process finished with exit code 0