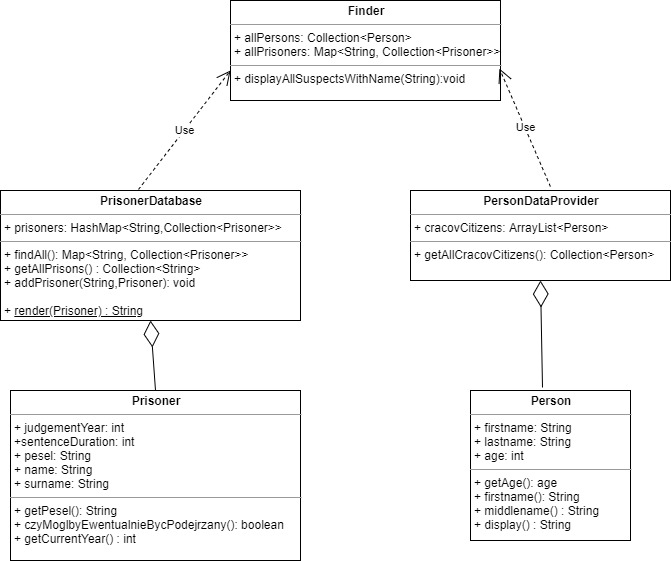
Raport PO – Refaktoryzacja

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;  
 }  
}

public class PersonDatabase {  
  
 private final Collection<Person> cracovPersons = 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> getAllCracovPersons() {  
 return cracovPersons;  
 }  
}

public class Finder {  
 private final Collection<Person> allPersons;  
  
 private final Map<String, Collection<Prisoner>> allPrisoners;  
  
 public Finder(Collection<Person> allPersons, Map<String, Collection<Prisoner>> allPrisoners) {  
 this.allPersons = allPersons;  
 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 : allPersons) {  
 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

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();  
}

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;  
 }  
}

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*);  
 }  
  
}

1. Dodanie iteratora

public interface SuspectAggregate {  
 Iterator<Suspect> iterator();  
 void generateData();  
}

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("There is no element");  
 }  
}

public class PersonDatabase implements SuspectAggregate {  
  
 private final Collection<Person> cracovPersons = new ArrayList<Person>();  
  
 public PersonDatabase() {}  
  
 public void generateData() {  
 addCracovPerson("Krzysztof", "Mendel", 30);  
 addCracovPerson("Pawel", "Pawlowicz", 32);  
 addCracovPerson("Janusz", "Programista", 77);  
 addCracovPerson("Alicja", "Zaczarowana", 22);  
 addCracovPerson("Janusz", "Gimbus", 15);  
 addCracovPerson("Tomek", "Gimbus", 14);  
 addCracovPerson("Piotr", "Zgredek", 29);  
 addCracovPerson("Kasia", "Kosinska", 19);  
 addCracovPerson("Janusz", "Mlodociany", 10);  
 addCracovPerson("Janusz", "Krakowski", 30);  
 addCracovPerson("Jan", "Kowalski", 30);  
 }  
  
 public void addCracovPerson(String firstName, String lastName, int age){  
 cracovPersons.add(new Person(firstName,lastName,age));  
 }  
  
 public Collection<Person> getAllCracovPersons() {  
 return cracovPersons;  
 }  
  
 @Override  
 public Iterator<Suspect> iterator(){  
 return new SuspectIterator(cracovPersons.iterator());  
 }  
}

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());  
 }  
}

public class Finder {  
 private final SuspectAggregate allPersons;  
  
 private final SuspectAggregate allPrisoners;  
  
 public Finder(PersonDatabase personDatabase, PrisonersDatabase prisonersDatabase) {  
 this.allPersons = personDatabase;  
 this.allPrisoners = prisonersDatabase;  
 }  
  
 public void displayAllSuspectsWithName(String name) {  
 ArrayList<Suspect> suspectedPersons = new ArrayList<Suspect>();  
 Iterator<? extends Suspect> prisonersIterator = allPrisoners.iterator();  
 Iterator<? extends Suspect> personsIterator = allPersons.iterator();  
  
 Suspect tempSuspect = null;  
 while(prisonersIterator.hasNext()){  
 tempSuspect = prisonersIterator.next();  
 if(tempSuspect.getFirstName().equals(name)&& tempSuspect.canBeSuspected()){  
 suspectedPersons.add(tempSuspect);  
 if(suspectedPersons.size() >=10) break;  
 }  
 }  
  
 if (suspectedPersons.size() < 10) {  
 while (personsIterator.hasNext()){  
 tempSuspect = personsIterator.next();  
 if (tempSuspect.canBeSuspected() && tempSuspect.getFirstName().equals(name)) {  
 suspectedPersons.add(tempSuspect);  
 }  
 if (suspectedPersons.size() >= 10) {  
 break;  
 }  
 }  
 }  
  
 System.*out*.println("Znalazlem " + suspectedPersons.size() + " pasujacych podejrzanych!");  
 for (Suspect sus:suspectedPersons) {  
 System.*out*.println(sus.toString());  
 }  
 }  
}

1. Dodanie kompozytu agregującego

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();  
 }  
  
  
}

zmiana w klasie Finder

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("Znalazlem " + suspectPeople.size() + " pasujacych podejrzanych!");  
 for (Suspect sus:suspectPeople) {  
 System.*out*.println(sus.toString());  
 }  
 }  
}

1. Dodanie opcji filtrowania i klasy Student

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) {  
 return suspect.getFirstName().equals(name);  
 }  
}

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;  
 }  
}

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));  
 }  
}

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("There is no element");  
 }  
}

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

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(SearchStrategy searchStrategy) {  
 return new SuspectIterator(prisoners  
 .values()  
 .stream()  
 .flatMap(Collection::stream)  
 .collect(Collectors.*toList*())  
 .iterator(),searchStrategy);  
 }  
}