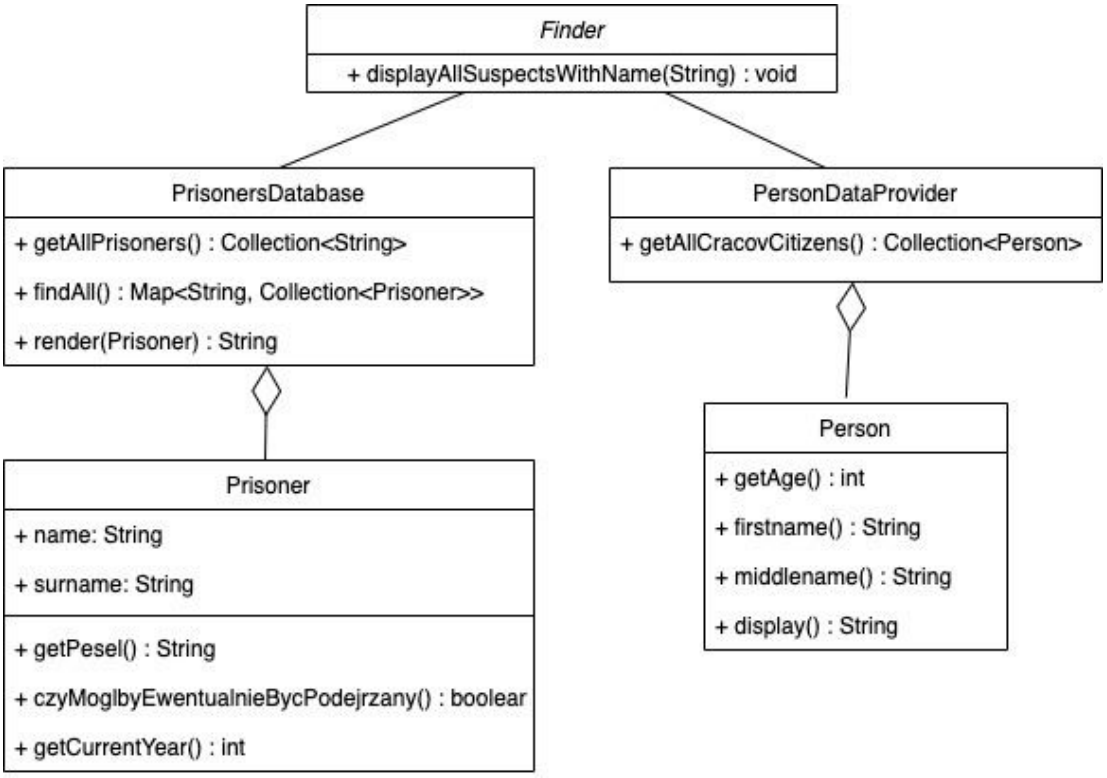
Nazar Kordiumov
Laboratorium 4
Refaktoryzacja

1. **Krok 1**
    a. Diagram



    b. Proponowane zmiany
        i. Nazewnictwo: wszystkie nazwy metod i pól po angielsku. Zastosowanie patternu CamelCase.
        ii. Widoczność metod: stworzyć metody dostępowe (gettery) do każdej klasy
        iii. Wprowadzenie osobnej klasy Prison/PrisonCategory

2. **Krok 2 - Refaktoryzacja**
    a. Zmiana klasy PrisonersDatabase

```java
public Map<String, Collection<Prisoner>> findAll() { return prisoners; }

private void addPrisoner(String category, Prisoner prisoner) {
    prisoners.putIfAbsent(category, new ArrayList<>());
    prisoners.get(category).add(prisoner);
}
```

b. Zmiana klasy Prisoner

```java
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 boolean IsJailedNow() {
        return judgementYear + sentenceDuration >= getCurrentYear();
    }

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

    private int getCurrentYear() {
        return LocalDate.now().getYear();
    }
}
```

c. Zmiana klasy Person

```java
public class CracovCitizen {

    private final String name;
    private final String surname;
    private final int age;

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

    public String getName() { return name; }

    public String getSurname() { return surname; }

    public int getAge() { return age; }

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

3. Krok 3 - Generalizacja klas Person i Prisoner

a. Użyję klasy abstrakcyjnej jako uogólnienia klas Prisoner oraz CracovCitizen. Klasy abstrakcyjne pozwalają na definiowanie pól prywatnych w porównaniu do interfejsów, w których każde pole jest public static final (constant).

```
public abstract class Suspect {
    private final String name;
    private final 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 getName() + " " + getSurname();
    }

    public abstract boolean canBeAccused();
}
```

b. Dodałem prywatne pola name i surname oraz metody, które były wspólne dla klas Prisoner i CracovCitizen czyli: getName(), getSurname(), display()

c. Dodatkowo wprowadziłem metodę canBeAccused(), która pozwala w małym stopniu uogólnić klasę Finder (metodę displayAllSuspectsWithName())

```java
    public void displayAllSuspectsWithName(String name) {
        List<Suspect> suspects = new ArrayList<>();

        for (Collection<Prisoner> prisonerCollection : allPrisoners.values()) {
            for (Prisoner prisoner : prisonerCollection) {
                if (isAppropriateSuspect(prisoner, name)) {
                    suspects.add(prisoner);
                }
                if (suspects.size() >= 10) {
                    break;
                }
            }
            if (suspects.size() >= 10) {
                break;
            }
        }

        if (suspects.size() < 10) {
            for (CracovCitizen cracovCitizen : allCracovCitizens) {
                if (isAppropriateSuspect(cracovCitizen, name)) {
                    suspects.add(cracovCitizen);
                }
                if (suspects.size() >= 10) {
                    break;
                }
            }
        }

        System.out.println("Znalazlem " + suspects.size() + " pasujacych podejrzanych!");
        suspects.forEach(suspect -> System.out.println(suspect.display()));
    }

    private boolean isAppropriateSuspect(Suspect suspect, String name) {
        return suspect.canBeAccused() && suspect.getName().equals(name);
    }
}
```

4. Krok 4 - Dodanie Iteratora

a. Stworzyłem interfejs SuspectAggregate

```java
public interface SuspectAggregate {
    Iterator<Suspect> iterator(String name);
}
```

b. Stworzyłem własny iterator SuspectIterator, który przyjmuje iterator kolekcji zawierającej obiekty klasy Suspect albo jej klasy dziedziczące

```java
public class SuspectIterator implements Iterator<Suspect> {

    Suspect suspect;
    Iterator<? extends Suspect> iterator;
    String name;

    public SuspectIterator(Iterator<? extends Suspect> iterator, String name) {
        this.iterator = iterator;
        this.name = name;
    }

    @Override
    public boolean hasNext() {
        while(iterator.hasNext()) {
            Suspect tempSuspect = iterator.next();
            if(tempSuspect.getName().equals(name) && tempSuspect.canBeAccused()) {
                suspect = tempSuspect;
                return true;
            }
        }
        return false;
    }

    @Override
    public Suspect next() {
        if(suspect != null) {
            return suspect;
        }
        throw new NoSuchElementException("There is no more suspects match the predicate");
    }
```

c. Następnie zrobiłem aby klasy PersonDataProvider oraz PrisonersDatabase implementowały interfejs SuspectAggregate

i. W klasie PersonDataProvider zwracam iterator kolekcji cracovCitizents

```java
@Override
public Iterator<Suspect> iterator(String name) {
    return new SuspectIterator(cracovCitizens.iterator(), name);
}
```

ii. W klasie PrisonersDatabase zwracam iterator kolekcji wszystkich Prisoner’ów znajdujących się w mapie

```java
@Override
public Iterator<Suspect> iterator(String name) {
    Iterator<Prisoner> prisonerIterator = prisoners.values().stream() Stream<Collection<Prisoner>>
            .flatMap(Collection::stream) Stream<Prisoner>
            .collect(Collectors.toList()) List<Prisoner>
            .iterator();
    return new SuspectIterator(prisonerIterator, name);
}
```

d. Ostatecznie klasa Finder wygląda następująco (znacznie lepiej)

```java
public class Finder {

    private final SuspectAggregate allCracovCitizens;
    private final SuspectAggregate allPrisoners;

    public Finder(PersonDataProvider personDataProvider, PrisonersDatabase prisonersDatabase) {
        this.allCracovCitizens = personDataProvider;
        this.allPrisoners = prisonersDatabase;
    }

    public void displayAllSuspectsWithName(String name) {
        Iterator<? extends Suspect> prisonersIterator = allPrisoners.iterator(name);
        Iterator<? extends Suspect> cracovCitizensIterator = allCracovCitizens.iterator(name);
        List<Suspect> suspects = new ArrayList<>();

        while(prisonersIterator.hasNext() && suspects.size() < 10) {
            suspects.add(prisonersIterator.next());
        }

        while(cracovCitizensIterator.hasNext() && suspects.size() < 10) {
            suspects.add(cracovCitizensIterator.next());
        }

        System.out.println("Znalazlem " + suspects.size() + " pasujacych podejrzanych!");
        suspects.forEach(System.out::println);
    }
```

5. Krok 5 - Composite

a. Dodałem klasę CompositeAggregate, która implementuje interfejs SuspectAggregate. Główna metoda iterator(), łączy wszystkie osoby ze wszystkich zbiorów (iteratorów) w jeden zbiór i zwraca jego iterator

```java
public class CompositeAggregate implements SuspectAggregate {

    private final List<SuspectAggregate> aggregates;

    public CompositeAggregate(List<SuspectAggregate> aggregates) {
        this.aggregates = aggregates;
    }

    @Override
    public Iterator<Suspect> iterator(String name) {
        Collection<Suspect> suspects = new ArrayList<>();
        aggregates.forEach(sA -> {
            Iterator<Suspect> iterator = sA.iterator(name);
            while(iterator.hasNext()) {
                suspects.add(iterator.next());
            }
        });
        return suspects.iterator();
    }
}
```

b. Dzięki temu usprawnieniu klasa Finder pozbyła kolejnej odpowiedzialności

```java
public class Finder {

    private final CompositeAggregate compositeAggregate;

    public Finder(CompositeAggregate compositeAggregate) {
        this.compositeAggregate = compositeAggregate;
    }

    public void displayAllSuspectsWithName(String name) {
        Iterator<Suspect> suspectIterator = compositeAggregate.iterator(name);
        List<Suspect> suspects = new ArrayList<>();

        while(suspectIterator.hasNext() && suspects.size() < 10) {
            suspects.add(suspectIterator.next());
        }

        System.out.println("Znalazlem " + suspects.size() + " pasujacych podejrzanych!");
        suspects.forEach(System.out::println);
    }
}
```

6. Krok 6 - Predicates

a. Najpierw dodałem pole int age oraz metodę getAge() do klasy abstrakcyjnej Suspect

b. Zamiast tworzenia własnego interfejsu SearchStrategy, wykorzystałem wbudowany funkcyjny interfejs Predicate. Po czym stworzyłem klasę CompositeSearchStrategy

```java
public class CompositeSearchStrategy implements Predicate<Suspect> {

    private final List<Predicate<Suspect>> filters;

    public CompositeSearchStrategy(List<Predicate<Suspect>> filters) { this.filters = filters; }

    @Override
    public boolean test(Suspect suspect) {
        return filters.stream()
                .allMatch(suspectPredicate -> suspectPredicate.test(suspect));
    }
}
```

c. Zmieniłem interfejs SuspectAggregate tak aby metoda iterator() przejmowała Predykat oraz zmieniłem wywołania tej metody wszędzie gdzie była używana

```java
public interface SuspectAggregate {
    Iterator<Suspect> iterator(Predicate<Suspect> filter);
}
```

d. Dodałem dwie strategie wyszukiwania
   i. Po imieniu

```java
public class NameSearchStrategy implements Predicate<Suspect> {

    private final String name;

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

    @Override
    public boolean test(Suspect suspect) { return suspect.getName().equals(name); }
}
```

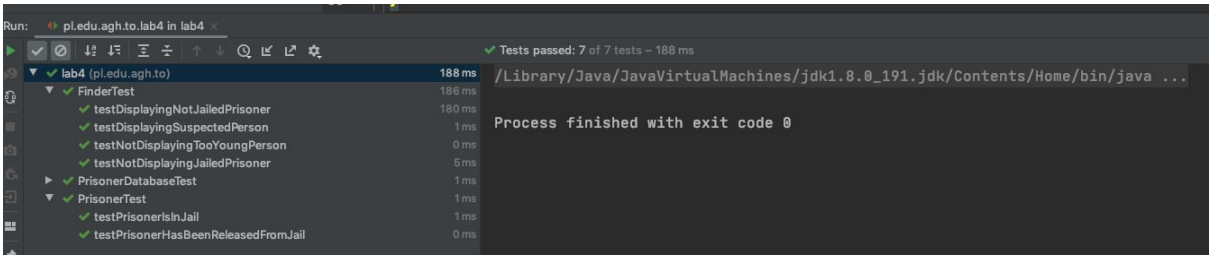
   ii. Po wieku

```java
public class AgeSearchStrategy implements Predicate<Suspect> {

    private final int age;

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

    @Override
    public boolean test(Suspect suspect) {
        return suspect.getAge() == age;
    }
}
```

e. Ostatecznie klasa Finder wygląda następująco:

```java
public class Finder {

    private final CompositeAggregate compositeAggregate;

    public Finder(CompositeAggregate compositeAggregate) { this.compositeAggregate = compositeAggregate; }

    public void display(CompositeSearchStrategy searchStrategy) {
        Iterator<Suspect> suspectIterator = compositeAggregate.iterator(searchStrategy);
        List<Suspect> suspects = new ArrayList<>();

        while(suspectIterator.hasNext() && suspects.size() < 10) {
            suspects.add(suspectIterator.next());
        }

        System.out.println("Znalazlem " + suspects.size() + " pasujacych podejrzanych!");
        suspects.forEach(System.out::println);
    }
}
```

f. Testy oraz widok klasy Application




```
public class Application {

    public static void main(String[] args) {
        PersonDataProvider personDataProvider = new PersonDataProvider();
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Jan",  surname: "Kowalski",  age: 30));
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Janusz",  surname: "Krakowski",  age: 30));
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Janusz",  surname: "Mlodociany",  age: 10));
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Kasia",  surname: "Kosinska",  age: 19));
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Piotr",  surname: "Zgredek",  age: 29));
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Tomek",  surname: "Gimbus",  age: 14));
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Janusz",  surname: "Gimbus",  age: 15));
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Alicja",  surname: "Zaczarowana",  age: 22));
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Janusz",  surname: "Programista",  age: 77));
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Pawel",  surname: "Pawlowicz",  age: 32));
        personDataProvider.getAllCracovCitizens().add(new CracovCitizen( name: "Krzysztof",  surname: "Mendel",  age: 30));
        PrisonersDatabase prisonersDatabase = new PrisonersDatabase();
        prisonersDatabase.addPrisoner( category: "Wiezienie krakowskie", new Prisoner( name: "Jan",  surname: "Kowalski",  age: 27, pesel: "87080452
        prisonersDatabase.addPrisoner( category: "Wiezienie krakowskie", new Prisoner( name: "Anita",  surname: "Wiercipieta",  age: 35, pesel: "840
        prisonersDatabase.addPrisoner( category: "Wiezienie krakowskie", new Prisoner( name: "Janusz",  surname: "Zlowieszczy",  age: 67, pesel: "92
        prisonersDatabase.addPrisoner( category: "Wiezienie przedmiejskie", new Prisoner( name: "Janusz",  surname: "Zamkniety",  age: 78,  pesel: "8
        prisonersDatabase.addPrisoner( category: "Wiezienie przedmiejskie", new Prisoner( name: "Adam",  surname: "Future",  age: 89, pesel: "880216
        prisonersDatabase.addPrisoner( category: "Wiezienie przedmiejskie", new Prisoner( name: "Zbigniew",  surname: "Nienajedzony",  age: 31,  pes
        prisonersDatabase.addPrisoner( category: "Wiezienie centralne", new Prisoner( name: "Jan",  surname: "Przedziwny",  age: 19,  pesel: "9110314
        prisonersDatabase.addPrisoner( category: "Wiezienie centralne", new Prisoner( name: "Janusz",  surname: "Podejrzany",  age: 52, pesel: "8512
        Finder suspects = new Finder(new CompositeAggregate(List.of(personDataProvider, prisonersDatabase)));
        suspects.display(new CompositeSearchStrategy(List.of(new NameSearchStrategy("Jan"), new AgeSearchStrategy(19))));
    }
```