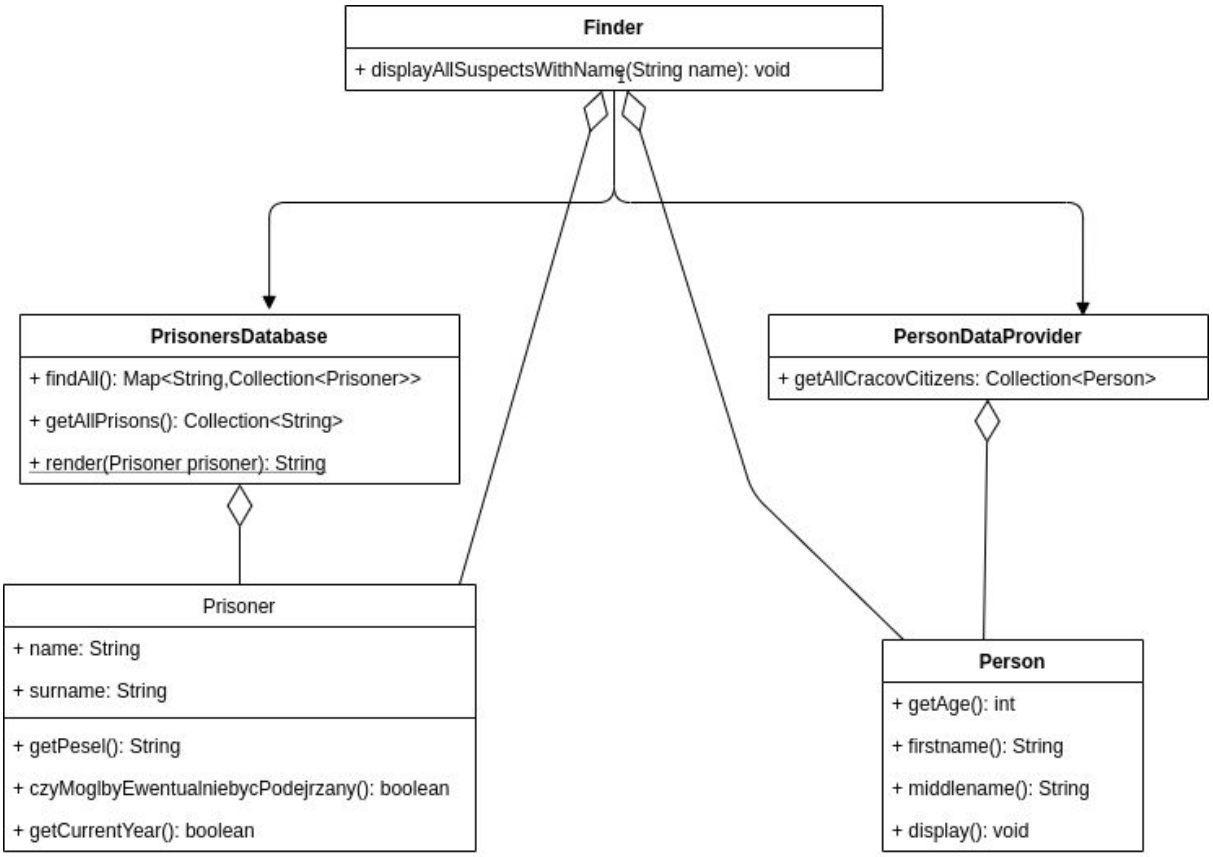
Projektowanie Obiektowe

Laboratorium 4

Konrad Siuzdak, Paweł Kocimski

Zadanie 1

Diagram narysowany na podstawie kodu:



Znaleziono następujące,rzucające się w oczy błędy:

- mieszanie języka polskiego i angielskiego w kodzie
- brak konwencji - używane zarówno pola publiczne(niefinalne) jaki i gettery
- złe nazwy metod, np. metoda firstname powinna się nazywać getFirstname, zgodnie z “konwencjami javy”
- Prisoner powinno mieć jakiś związek(dziedziczenie lub interfejs) z Person
- długie nazwy metod, np. czyMoglbyEwentualniebycPodejrzany
- render w klasie PrisonersDatabase jest niepotrzebnie statyczna, można to zrobić lepiej
- getCurrentYear() nie powinno być metodą klasy Prisoner, ponieważ zaburza to zasadę pojedynczej odpowiedzialności i nie ma związku z odwzorowaniem świata rzeczywistego w klasie
- brak pól finalnych, np imię osoby raczej się nie zmienia
- Finder niepotrzebnie tworzy nowe kolekcje Person i Prisoner

Zadanie 2

Kod został poprawiony według diagramu z zadania.

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

   private final int senteceDuration;

   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.senteceDuration = sentenceDuration;
   }

   public String getName() {
       return name;
   }

   public String getSurname() {
       return surname;
   }


   public boolean isJailedNow()
   {
       return judgementYear+senteceDuration>Calendar.getInstance().get(Calendar.YEAR);
   }
}

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;
    }
}
```

Zadanie 3

Dodanie interfejsu Suspect wydaje się poprawić czytelność kodu, jednak jeszcze lepszym rozwiązaniem będzie dodanie klasy abstrakcyjnej Suspect ze względu na to, że pola oraz implementacje pewnych metod w klasach Prisoner i CracovCitizen się powtarzają. Dodatkowo wprowadzono metodą canBeAccused, która umożliwia uogólnienie klasy Finder.

```
public abstract class Suspect {

    protected String name;
    protected String surname;

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }

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

    public abstract boolean canBeAccused();
}



public class Prisoner extends Suspect{
    private final int judgementYear;

    private final int senteceDuration;

    private final String pesel;

    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.senteceDuration = sentenceDuration;
    }


    public boolean isJailedNow()
    {
        return judgementYear+senteceDuration>Calendar.getInstance().get(Calendar.YEAR);
    }

    @Override
    public boolean canBeAccused() {
        return !isJailedNow();
    }
}

public class CracovCitizen extends Suspect {
    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;
    }

    @Override
    public boolean canBeAccused() {
        return this.age>18;
    }
}
```

Zadanie 4

Utworzono klasę FlatIterator, zmodyfikowano klasy dostarczające dane implementując interfejs SuspectAggregate

```
public class FlatIterator implements Iterator<Suspect> {

    private final Map<String, Collection<Prisoner>> prisoners;
    private final List<Iterator<Prisoner>> iteratorsOfPrisoners;
    private int index=0;

    public FlatIterator(Map<String, Collection<Prisoner>> prisoners) {
        this.prisoners = prisoners;
        iteratorsOfPrisoners=new ArrayList<>();
        createListOfIterators();


    }

    private void createListOfIterators()
    {
        for(Collection<Prisoner> prisonerCollection:prisoners.values())
```

```
        {
            iteratorsOfPrisoners.add(prisonerCollection.iterator());
        }
    }

    @Override
    public boolean hasNext() {
        if(index < iteratorsOfPrisoners.size()&&iteratorsOfPrisoners.get(index).hasNext()) return true;
        if(index+1 < iteratorsOfPrisoners.size()) return iteratorsOfPrisoners.get(index+1).hasNext();
        return false;
    }

    @Override
    public Suspect next() {
        if(iteratorsOfPrisoners.get(index).hasNext()) return iteratorsOfPrisoners.get(index).next();
        index++;
        return iteratorsOfPrisoners.get(index).next();
    }
}

public interface SuspectAggregate {
    Iterator<? extends Suspect> iterator();
}
public class PrisonersDatabase implements SuspectAggregate {
         ….
    @Override
    public Iterator<? extends Suspect> iterator() {
        return new FlatIterator(prisoners);
    }

}

public class PersonDataProvider implements SuspectAggregate {
         …..
    @Override
    public Iterator<? extends Suspect> iterator() {
        return this.cracovCitizens.iterator();
    }
}
```

Zmodyfikowana klasa Finder korzystająca z iteratorów

```
public class Finder {
    private final SuspectAggregate prisonerDatabase;
    private final SuspectAggregate personDataProvider;

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


    public void displayAllSuspectsWithName(String name) {
        ArrayList<Suspect> suspected= new ArrayList<Suspect>();
        Iterator<? extends Suspect> prisonerIterator = prisonerDatabase.iterator();
        Iterator<? extends Suspect> personIterator = personDataProvider.iterator();

        Suspect suspect = null;
        while(personIterator.hasNext()) {
                suspect = personIterator.next();
                if (suspect.canBeAccused() && suspect.getName().equals(name)) {
```

```
                    suspected.add(suspect);
                }
                if (suspected.size() >= 10) {
                    break;
                }
            }


        if (suspected.size() < 10) {
            while (prisonerIterator.hasNext()) {
                suspect = prisonerIterator.next();
                if (suspect.canBeAccused() && suspect.getName().equals(name)) {
                    suspected.add(suspect);
                }
                if (suspected.size()  >= 10) {
                    break;
                }
            }
        }

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

        for (Suspect suspectIn : suspected) {

                System.out.println(suspectIn.display());
        }

    }
}
```

Zadanie 5

Powstały w ten sposób program nie rozwiązuje problemu dodania do bazy danych nowego zbioru danych.Aby rozwiązać ten problem została stworzona klasa pośrednia przechowująca wszystkie bazy danych o podejrzanych. Dodatkowo stworzono iterator, który agreguje, przebiega po każdej bazie danych i umożliwia dostęp do każdego podejrzanego. Dzięki temu klasa Finder nie musi mieć wiedzy o poszczególnych bazach danych.

Pośrednicząca klasa CompositeAggregate

```
public class CompositeAggregate implements SuspectAggregate{
  private final List<SuspectAggregate> databases = new ArrayList<SuspectAggregate>() {};

  public CompositeAggregate() {

  }

  public void addDatabase(SuspectAggregate database){
     this.databases.add(database);
  }

  @Override
  public Iterator<? extends Suspect> iterator() {
     Collection<Suspect> suspects = new ArrayList<>();
```

```
            for(SuspectAggregate database : databases){
                Iterator<Suspect> iterator = (Iterator<Suspect>) database.iterator();
                while(iterator.hasNext()){
                    suspects.add(iterator.next());
                }
            }
            return suspects.iterator();
    }
}
```

Zmieniona klasa Finder

```
public class Finder {
    private final CompositeAggregate compositeAggregate;


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

    public void displayAllSuspectsWithName(String name) {
        ArrayList<Suspect> suspected= new ArrayList<Suspect>();
        Iterator<? extends Suspect> compositeIterator = compositeAggregate.iterator();


        Suspect suspect = null;
        while(compositeIterator.hasNext()) {
                suspect = compositeIterator.next();
                if (suspect.canBeAccused() && suspect.getName().equals(name)) {
                    suspected.add(suspect);
                }
                if (suspected.size() >= 10) {
                    break;
                }
            }


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

        for (Suspect suspectIn : suspected) {

                System.out.println(suspectIn.display());
        }

    }
}
```

Wywołanie programu

```
public class Application {

   public static void main(String[] args) {
      CompositeAggregate compositeAggregate = new CompositeAggregate();
      compositeAggregate.addDatabase(new PersonDataProvider());
      compositeAggregate.addDatabase(new PrisonersDatabase());
      Finder suspects = new Finder(compositeAggregate);
      suspects.displayAllSuspectsWithName("Janusz");
   }
}
```

Zadanie 5

Wprowadzono możliwość łatwego wyszukiwania, dzięki dodaniu SearchStrategy i implementujących go klas.

Interfejs SearchStrategy

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

Klasa AgeSearchStrategy

```
public class AgeSearchStrategy implements  SearchStrategy{

   private int age;

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

   @Override
   public boolean filter(Suspect suspect) {
      if(suspect instanceof CracovCitizen) {
         return ((CracovCitizen) suspect).getAge() > this.age;
      }
      return true;
   }
}
```

Klasa NameSearchStrategy

```
public class NameSearchStrategy implements SearchStrategy {
```

```
    private String name;

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

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

Klasa CompositeSearchStrategy

```
public class CompositeSearchStrategy implements SearchStrategy {

    List<SearchStrategy>searchStrategies = new ArrayList<>();

    public void addSearchStrategy(SearchStrategy searchStrategy){
        searchStrategies.add(searchStrategy);
    }

    @Override
    public boolean filter(Suspect suspect) {
        boolean result = true;
        for(SearchStrategy searchStrategy : searchStrategies){
            result = result && searchStrategy.filter(suspect);
        }
        return result;
    }
}
```

Następnie dodano bazę studentów i przetestowano aplikację

Klasa Student

```
public class Student extends Suspect {

    private String index;

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

    @Override
    public boolean canBeAccused() {
        return true;
    }
}
```

Klasa StudentDataProvider

```
public class StudentDataProvider implements SuspectAggregate{
  private final Collection<Student> Students = new ArrayList<>();

  public StudentDataProvider() {
     Students.add(new Student("Franciszek", "Nowak", "004040"));
     Students.add(new Student("Władysław", "Wiśniewski", "104041"));
     Students.add(new Student("Jeremi", "Wójcik", "204042"));
     Students.add(new Student("Zygmunt", "Koptyra", "304043"));
     Students.add(new Student("Karol", "Kreol", "404044"));
     Students.add(new Student("Janusz", "Karłowski", "504045"));
     Students.add(new Student("Augustyn", "Babinicz", "604046"));
     Students.add(new Student("Ambroży", "Billewicz", "704047"));

  }
  @Override
  public Iterator<? extends Suspect> iterator() {
     return this.Students.iterator();
  }
}
```

Zmieniona klasa Finder korzystająca z SearchStrategy

```
public class Finder {
  private final CompositeAggregate compositeAggregate;


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

  public void displayAllSuspectsWithName(CompositeSearchStrategy filters) {
     ArrayList<Suspect> suspected= new ArrayList<Suspect>();
     Iterator<? extends Suspect> compositeIterator = compositeAggregate.iterator();


     Suspect suspect = null;
     while(compositeIterator.hasNext()) {
           suspect = compositeIterator.next();
           if (filters.filter(suspect)) {
              suspected.add(suspect);
           }
           if (suspected.size() >= 10) {
              break;
           }
        }


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

     for (Suspect suspectIn : suspected) {

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

```
  }
}
```

Klasa Application

```
public class Application {

  public static void main(String[] args) {

    CompositeAggregate compositeAggregate = new CompositeAggregate();
    compositeAggregate.addDatabase(new PersonDataProvider());
    compositeAggregate.addDatabase(new PrisonersDatabase());
    compositeAggregate.addDatabase(new StudentDataProvider());
    Finder suspects = new Finder(compositeAggregate);

    CompositeSearchStrategy filters = new CompositeSearchStrategy();
    filters.addSearchStrategy(new NameSearchStrategy("Janusz"));
    filters.addSearchStrategy(new AgeSearchStrategy(10));

    suspects.displayAllSuspectsWithName(filters);
  }
}
```