

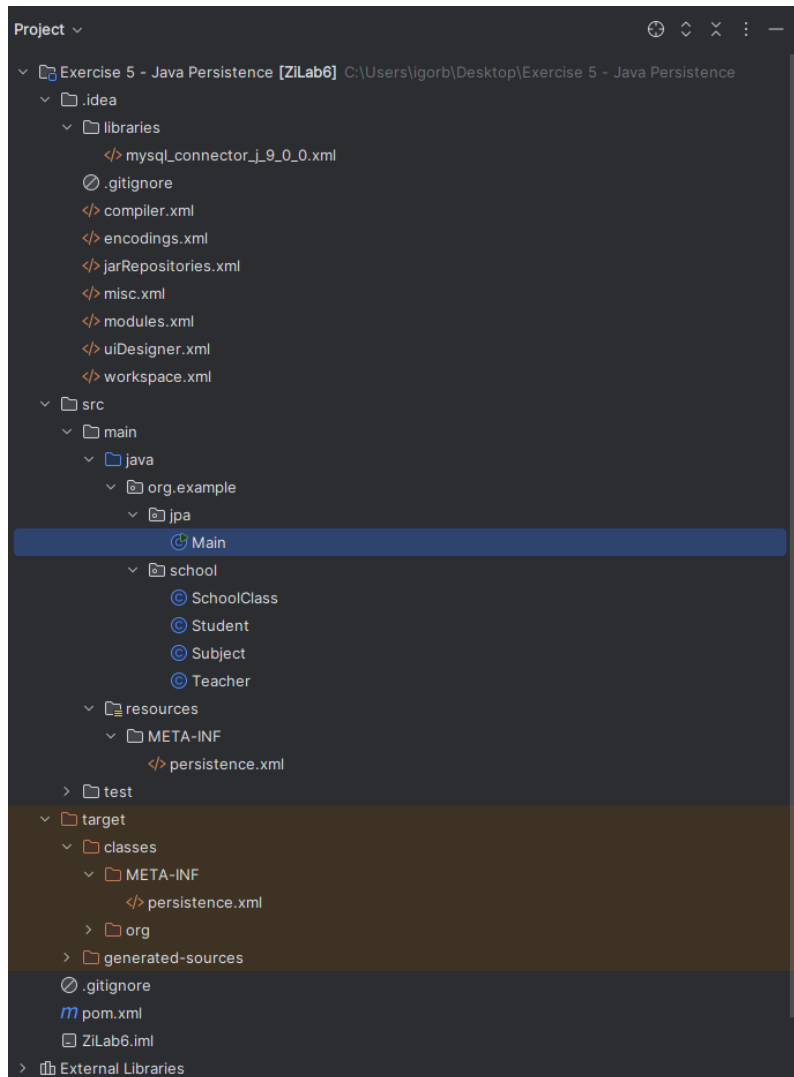
Sprawozdanie z Laboratorium 6

Igor Bębenek

Grupa 332

1. Struktura projektu

Stworzyłem pakiet `school` dla klas `Student`, `Teacher`, `SchoolClass` i `Subject` żeby wszystkie klasy móc podpiąć jako `import org.example.school.*`



Klasa uczeń

```
1 package org.example.school;
2
3 import jakarta.persistence.*;
4
5 @Entity 12 usages
6 public class Student {
7     @Id
8     @GeneratedValue(strategy = GenerationType.AUTO)
9     private Long id;
10
11     private String firstName; 2 usages
12     private String lastName; 2 usages
13
14     @ManyToOne 2 usages
15     private SchoolClass schoolClass;
16
17     public Student() {}
18
19     public Student(String firstName, String lastName, SchoolClass schoolClass) { 8 usages
20         this.firstName = firstName;
21         this.lastName = lastName;
22         this.schoolClass = schoolClass;
23     }
24
25     public Long getId() { no usages
26         return id;
27     }
28
29     public String getFirstName() { no usages
30         return firstName;
31     }
32
33     public String getLastName() { no usages
34         return lastName;
35     }
36
37     public SchoolClass getSchoolClass() { no usages
38         return schoolClass;
39     }
40 }
```

W klasie student przechowujemy dane ucznia – id, firstName, lastName.

@ManyToOne

private SchoolClass schoolClass; - każdy uczeń przynależy do dokładnie jednej klasy, wielu uczniów może należeć do jednej klasy

Potem stworzyłem pusty konstruktor domyślny żeby framework mógł tworzyć obiekty oraz konstruktor z parametrami

firstName, lastName, schoolClass dzięki któremu utworzymy obiekt ucznia z imieniem, nazwiskiem i klasą. Na końcu mamy gettery i setery umożliwiające odczyt i zapis wartości pól.

Klasa Nauczyciel

```
6 @Entity 14 usages
7 public class Teacher {
8     @Id
9     @GeneratedValue(strategy = GenerationType.AUTO)
10    private Long id;
11    private String firstName; 2 usages
12    private String lastName; 2 usages
13
14    @OneToOne(mappedBy = "teacher", cascade = CascadeType.ALL) 1 usage
15    private SchoolClass schoolClass;
16
17    @OneToMany(mappedBy = "teacher", cascade = CascadeType.ALL) 1 usage
18    private List<Subject> subjects = new ArrayList<>();
19
20    public Teacher() {}
21
22    public Teacher(String firstName, String lastName) { 3 usages
23        this.firstName = firstName;
24        this.lastName = lastName;
25    }
26
27    public Long getId() { no usages
28        return id;
29    }
30
31    public String getFirstName() { no usages
32        return firstName;
33    }
34
35    public String getLastName() { no usages
36        return lastName;
37    }
38
39    public SchoolClass getSchoolClass() { no usages
40        return schoolClass;
41    }
42
43    public List<Subject> getSubjects() { no usages
44        return subjects;
45    }
46 }
```

W klasie Teacher zawarliśmy pola id, firstName, lastName. Relacja OneToOne określa, że nauczyciel może pełnić funkcję

wychowawcy jednej klasy. **CascadeType.ALL** pozwala na automatyczne zapisywanie zmian w związanej klasie. **Mappedby** wskazuje w jakiej encji zdefiniowana jest relacja i, że znajduje się tam klucz obcy.

Relacja **OneToMany** określa relacje jeden do wielu z encją **Subject** – jeden nauczyciel może prowadzić wiele przedmiotów.

Następnie analogicznie jak w klasie **Student** mamy pusty konstruktor domyślny, konstruktor inicjalizujący dane osobowe oraz gettery i setery umożliwiające manipulacje danymi.

Klasa „klasa”

```

6      @Entity 12 usages
7      public class SchoolClass {
8          @Id
9          @GeneratedValue(strategy = GenerationType.AUTO)
10         private Long id;
11
12         private String name; 2 usages
13         private int level; 2 usages
14
15         @OneToMany(mappedBy = "schoolClass", cascade = CascadeType.ALL) 1 usage
16         private List<Student> students = new ArrayList<>();
17
18         @OneToOne(cascade = CascadeType.ALL) 2 usages
19         private Teacher teacher;
20
21         @ManyToMany(cascade = CascadeType.ALL) 1 usage
22         private List<Subject> subjects = new ArrayList<>();
23
24     public SchoolClass() {}
25     public SchoolClass(String name, int level, Teacher teacher) { 2 usages
26         this.name = name;
27         this.level = level;
28         this.teacher = teacher;
29     }
30
31     public Long getId() { no usages
32         return id;
33     }
34
35     public String getName() { no usages
36         return name;
37     }
38
39     public int getLevel() { no usages
40         return level;
41     }
42
43     public List<Student> getStudents() { no usages
44         return students;

```

```

43     public List<Student> getStudents() { no usages
44         return students;
45     }
46
47     public Teacher getTeacher() { no usages
48         return teacher;
49     }
50
51     public List<Subject> getSubjects() { 5 usages
52         return subjects;
53     }
54 }
55

```

W klasie SchoolClass mamy pola id, name, level – rok nauczania. Relacja OneToMany określa, że jedna klasa ma wielu uczniów, jest zarządzana przez pole schoolClass w klasie Student.

Relacja OneToOne określa, że każda klasa ma dokładnie jednego wychowawcę.

Relacja ManyToMany określa, że jedna klasa może mieć wiele przedmiotów, a jeden przedmiot realizowany być może przez wiele klas.

Następnie mamy analogicznie jak w poprzednich klasach, konstruktor domyślny, argumentowy, gettery oraz setery.

Klasa Przedmiot

```

4      import java.util.ArrayList;
5      import java.util.List;
6
7      @Entity 11 usages
8      public class Subject {
9          @Id
10         @GeneratedValue(strategy = GenerationType.AUTO)
11         private Long id;
12
13         private String name; 2 usages
14
15         @ManyToOne(cascade = CascadeType.ALL) 2 usages
16         private Teacher teacher;
17
18         @ManyToMany(mappedBy = "subjects") 1 usage
19         private List<SchoolClass> schoolClasses = new ArrayList<>();
20
21         public Subject() {}
22
23         public Subject(String name, Teacher teacher) { 3 usages
24             this.name = name;
25             this.teacher = teacher;
26         }
27
28         public Long getId() { no usages
29             return id;
30         }
31
32         public String getName() { no usages
33             return name;
34         }
35
36         public Teacher getTeacher() { no usages
37             return teacher;
38         }
39
40         public List<SchoolClass> getSchoolClasses() { no usages
41             return schoolClasses;
42         }
43     }

```

W klasie Subject mamy pola id, name.

Relacja ManyToOne określa, że jeden nauczyciel może mieć wiele przedmiotów, ale każdy przedmiot ma jednego nauczyciela.

Relacja ManyToMany określa, że jeden przedmiot może być realizowany przez wiele klas, a jedna klasa może mieć wiele przedmiotów.

Potem mamy konstruktory bezargumentowy, argumentowy , gettery i setery.

Metoda Main


```

Main.java x .gitignore pom.xml (ZiLab6) </> persistence.xml Student.java Teacher.java SchoolClass
1 package org.example.jpa;
2 import jakarta.persistence.EntityManager;
3 import jakarta.persistence.EntityManagerFactory;
4 import jakarta.persistence.Persistence;
5 import org.example.school.*;
6
7 public class Main {
8     public static void main(String[] args) {
9         EntityManagerFactory emf = Persistence.createEntityManagerFactory(persistenceUnitName: "schoolPU");
10        EntityManager em = emf.createEntityManager();
11
12        em.getTransaction().begin();
13
14        Teacher teacher1 = new Teacher("Anna", "Nowak");
15        Teacher teacher2 = new Teacher("Jan", "Kowalski");
16        Teacher teacher3 = new Teacher("Maria", "Wiśniewska");
17
18        Subject math = new Subject(name: "Matematyka", teacher1);
19        Subject physics = new Subject(name: "Fizyka", teacher2);
20        Subject biology = new Subject(name: "Biologia", teacher3);
21
22        SchoolClass class1A = new SchoolClass(name: "Klasa 1A", level: 1, teacher1);
23        class1A.getSubjects().add(math);
24        class1A.getSubjects().add(physics);
25        class1A.getSubjects().add(biology);
26
27        SchoolClass class2B = new SchoolClass(name: "Klasa 2B", level: 2, teacher2);
28        class2B.getSubjects().add(math);
29        class2B.getSubjects().add(biology);
30
31        em.persist(new Student(firstName: "Krzysztof", lastName: "Nowicki", class1A));
32        em.persist(new Student(firstName: "Agnieszka", lastName: "Kowal", class1A));
33        em.persist(new Student(firstName: "Tomasz", lastName: "Zieliński", class1A));
34        em.persist(new Student(firstName: "Joanna", lastName: "Mazur", class1A));
35
36        em.persist(new Student(firstName: "Paweł", lastName: "Lewandowski", class2B));
37        em.persist(new Student(firstName: "Ewa", lastName: "Kamińska", class2B));
38        em.persist(new Student(firstName: "Zofia", lastName: "Wiśniewska", class2B));
39        em.persist(new Student(firstName: "Marcin", lastName: "Dąbrowski", class2B));
40
41        em.persist(teacher1);

```

```

39        em.persist(new Student(firstName: "Marcin", lastName: "Dąbrowski", class2B));
40
41        em.persist(teacher1);
42        em.persist(teacher2);
43        em.persist(teacher3);
44        em.persist(math);
45        em.persist(physics);
46        em.persist(biology);
47        em.persist(class1A);
48        em.persist(class2B);
49
50        em.getTransaction().commit();
51
52        System.out.println("\nPrzedmioty ucznia:");
53        em.createQuery(s: """
54        SELECT s.firstName, s.lastName, sub.name, t.firstName, t.lastName FROM Student s JOIN s.schoolClass sc JOIN sc.subjects sub JOIN sub.teacher t WHERE s.id = :studentId
55        """, Object[].class).setParameter(s: "studentId", @ 1L)
56        .getResultList().forEach(result -> System.out.println("Uczeń: " + result[0] + " " + result[1] + ", Przedmiot: " + result[2] + ", Nauczyciel: " + result[3] + " " + result[4]));
57
58        System.out.println("\nUczniowie nauczyciela:");
59        em.createQuery(s: """
60        SELECT t.firstName, t.lastName, s.firstName, s.lastName, sc.name FROM Teacher t JOIN t.subjects sub JOIN sub.schoolClasses sc JOIN sc.students s WHERE t.id = :teacherId
61        """, Object[].class).setParameter(s: "teacherId", @ 1L).getResultList()
62        .forEach(result -> System.out.println("Nauczyciel: " + result[0] + " " + result[1] + ", Uczeń: " + result[2] + " " + result[3] + ", Klasa: " + result[4]));
63
64        em.close();
65        emf.close();
66    }
67 }

```

Tak jak na poprzednich laboratoriach tworzymy fabrykę menedżera encji i menedżera encji. Tworzymy trzech przykładowych nauczycieli oraz 3 przedmioty.

Tworzymy klasy o poziomie nauczania, wychowawcy i przypisujemy im przedmioty (np. class1A ma poziom 1, teacher1 jest wychowawcą, ma przedmioty math, physics, biology).

Każda klasa ma przypisanych czterech uczniów typu Student.

Wszystkie tworzenia obiektów są realizowane za pomocą jednej transakcji.

Wczytanie i wyświetlenie danych ucznia

```
System.out.println("\nPrzedmioty ucznia:");
em.createQuery("SELECT s.firstName, s.lastName, sub.name, t.firstName, t.lastName FROM Student s JOIN s.schoolClass sc JOIN sc.subjects sub JOIN sub.teacher t WHERE s.id = :studentId", Object[].class).setParameter("studentId", 1L).getResultList().forEach(result -> System.out.println("Uczeń: " + result[0] + " " + result[1] + ", Przedmiot: " + result[2] + ", Nauczyciel: " + result[3] + " " + result[4]));
```

Wczytanie i wyświetlenie danych nauczyciela oraz jego uczniów z jego przedmiotów

```
System.out.println("\nUczniowie nauczyciela:");
em.createQuery("SELECT t.firstName, t.lastName, s.firstName, s.lastName, sc.name FROM Teacher t JOIN t.subjects sub JOIN sub.schoolClasses sc JOIN sc.students s WHERE t.id = :teacherId", Object[].class).setParameter("teacherId", 1L).getResultList().forEach(result -> System.out.println("Nauczyciel: " + result[0] + " " + result[1] + ", Uczeń: " + result[2] + " " + result[3] + ", Klasa: " + result[4]));
```

Wyniki z konsoli:

```
Przedmioty ucznia:
Uczeń: Krzysztof Nowicki, Przedmiot: Matematyka, Nauczyciel: Anna Nowak
Uczeń: Krzysztof Nowicki, Przedmiot: Fizyka, Nauczyciel: Jan Kowalski
Uczeń: Krzysztof Nowicki, Przedmiot: Biologia, Nauczyciel: Maria Wiśniewska

Uczniowie nauczyciela:
Nauczyciel: Anna Nowak, Uczeń: Krzysztof Nowicki, Klasa: Klasa 1A
Nauczyciel: Anna Nowak, Uczeń: Agnieszka Kowal, Klasa: Klasa 1A
Nauczyciel: Anna Nowak, Uczeń: Tomasz Zieliński, Klasa: Klasa 1A
Nauczyciel: Anna Nowak, Uczeń: Joanna Mazur, Klasa: Klasa 1A
Nauczyciel: Anna Nowak, Uczeń: Paweł Lewandowski, Klasa: Klasa 2B
Nauczyciel: Anna Nowak, Uczeń: Ewa Kamińska, Klasa: Klasa 2B
Nauczyciel: Anna Nowak, Uczeń: Zofia Wiśniewska, Klasa: Klasa 2B
Nauczyciel: Anna Nowak, Uczeń: Marcin Dąbrowski, Klasa: Klasa 2B
```

Tabele:

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> schoolclass	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	32.0 KiB	-
<input type="checkbox"/> schoolclass_seq	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> schoolclass_subject	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	48.0 KiB	-
<input type="checkbox"/> student	★ Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_general_ci	32.0 KiB	-
<input type="checkbox"/> student_seq	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> subject	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	32.0 KiB	-
<input type="checkbox"/> subject_seq	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> teacher	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	16.0 KiB	-
<input type="checkbox"/> teacher_seq	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
9 tables	Sum	25	InnoDB	utf8mb4_general_ci	224.0 KiB	0 B

✓ Showing rows 0 - 1 (2 total, Query took 0.0002 seconds.)

```
SELECT * FROM `schoolclass`
```

☐ Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

☐ Show all | Number of rows: 25 Filter rows: Search this table Sort

Extra options

			id	level	name	teacher_id
<input type="checkbox"/>	Edit	Copy	Delete	1	1 Klasa 1A	1
<input type="checkbox"/>	Edit	Copy	Delete	2	2 Klasa 2B	2

↑ ☐ Check all With selected: Edit Copy Delete Export

✓ Showing rows 0 - 4 (5 total, Query took 0.0002 seconds.)

`SELECT * FROM `schoolclass_subject``

☐ Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

☐ Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

schoolClasses_id	subjects_id
1	1
1	2
1	3
2	1
2	3

























✓ Showing rows 0 - 7 (8 total, Query took 0.0002 seconds.)

`SELECT * FROM `student``

☐ Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key

Extra options

<div><div><div></div><div></div><div></div></div></div>				id	firstName	lastName	schoolClass_id
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Krzysztof	Nowicki	1
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	Agnieszka	Kowal	1
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	Tomasz	Zieliński	1
<input type="checkbox"/>	 Edit	 Copy	 Delete	4	Joanna	Mazur	1
<input type="checkbox"/>	 Edit	 Copy	 Delete	5	Paweł	Lewandowski	2
<input type="checkbox"/>	 Edit	 Copy	 Delete	6	Ewa	Kamińska	2
<input type="checkbox"/>	 Edit	 Copy	 Delete	7	Zofia	Wiśniewska	2
<input type="checkbox"/>	 Edit	 Copy	 Delete	8	Marcin	Dąbrowski	2

 ☐ Check all | With selected:  Edit  Copy  Delete  Export

✓ Showing rows 0 - 2 (3 total, Query took 0.0002 seconds.)

```
SELECT * FROM `subject`
```

☐ Profiling [[Edit inline](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

☐ Show all | Number of rows: Filter rows:

Extra options

				id	name	teacher_id
<input type="checkbox"/>				1	Matematyka	1
<input type="checkbox"/>				2	Fizyka	2
<input type="checkbox"/>				3	Biologia	3

✓ Showing rows 0 - 2 (3 total, Query took 0.0002 seconds.)

```
SELECT * FROM `teacher`
```

☐ Profiling [[Edit inline](#)] [[Edit](#)] [[Explain SQL](#)] [[Create PHP code](#)] [[Refresh](#)]

☐ Show all | Number of rows: Filter rows:

Extra options

				id	firstName	lastName
<input type="checkbox"/>				1	Anna	Nowak
<input type="checkbox"/>				2	Jan	Kowalski
<input type="checkbox"/>				3	Maria	Wiśniewska