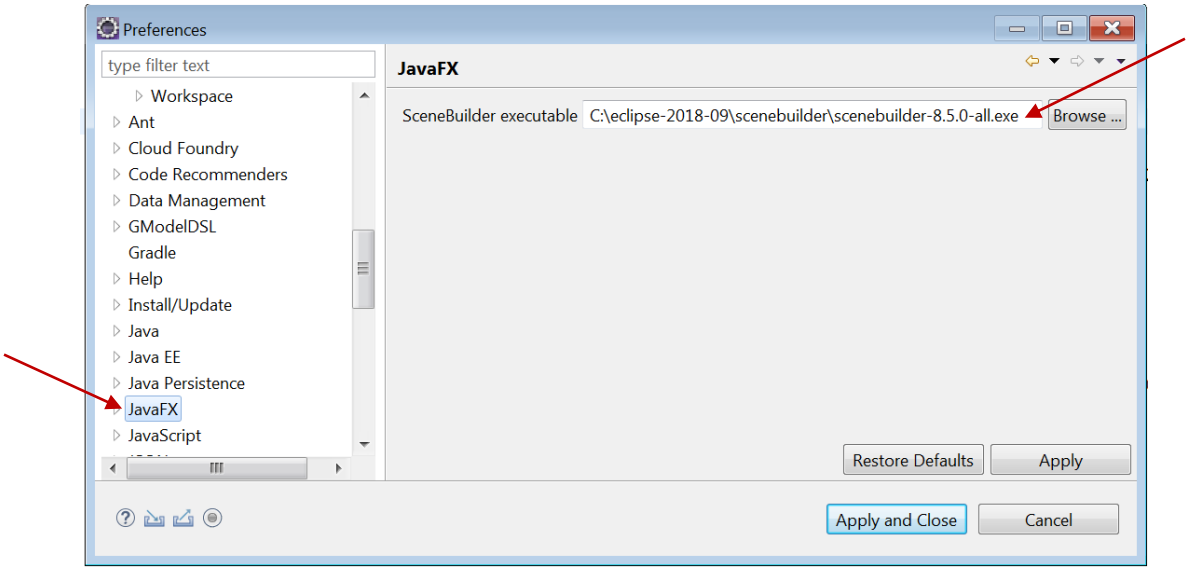
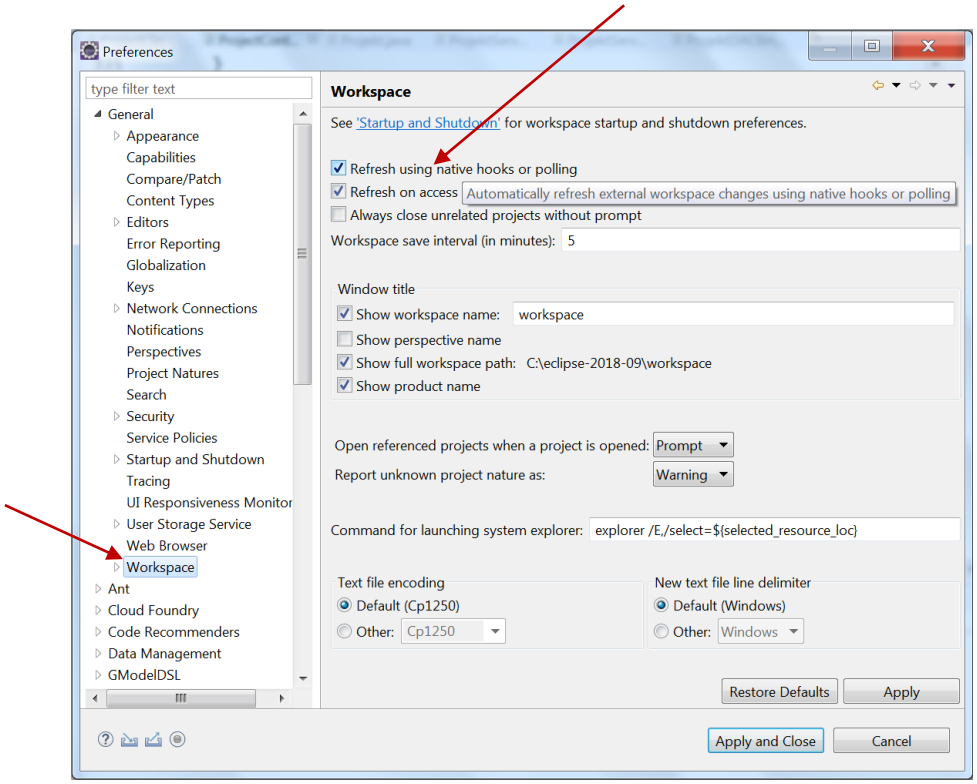
APLIKACJA BAZODANOWA (JAVAFX, JPA/HIBERNATE)

1. INSTALACJA

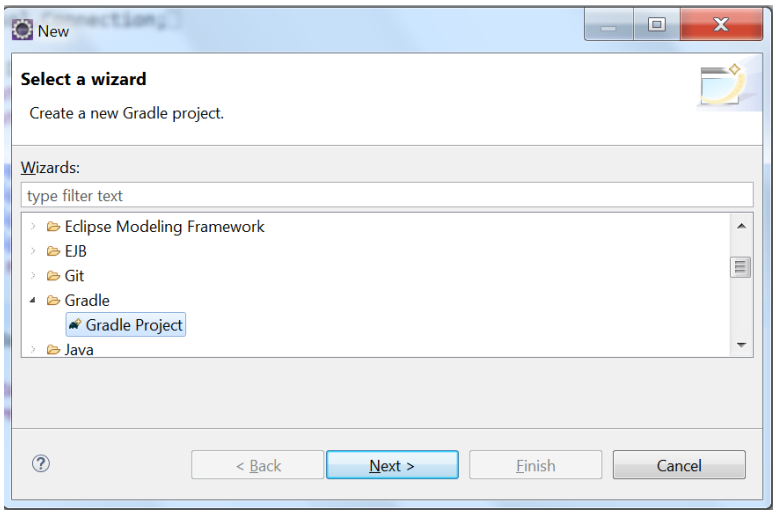
1.1. Ściągnij udostępnione archiwum https://drive.google.com/open?id=1UIi6HoKWUM0ohcojDGN1Z08Zl42UirNs (zawiera Eclipse, JDK 8, Scene Builder) i rozpakuj bezpośrednio na dysku C.
Po uruchomieniu Eclipse z menu wybierz *Window -> Preferences* i sprawdź zaznaczone poniżej ustawienia.



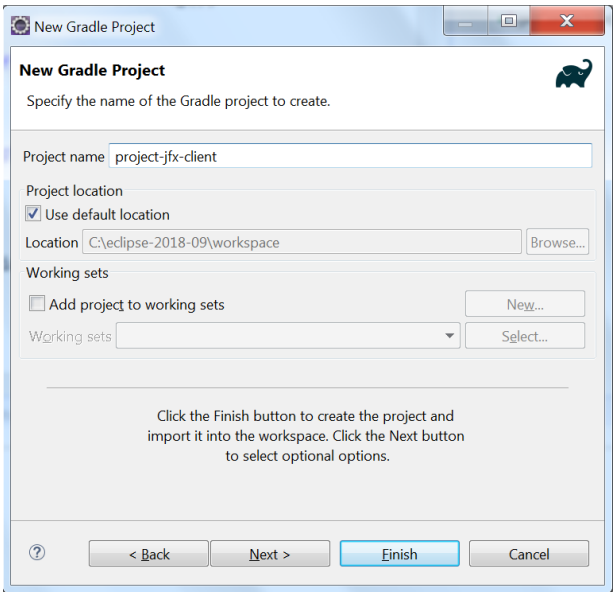


2. UTWORZENIE PROJEKTU

2.1. Wybierz z menu *File -> New -> Other* (lub użyj skrótu *CTRL + N*), następnie zaznacz *Gradle -> Gradle Project* i kliknij *Next*. W nowo otwartym oknie kliknij ponownie *Next*.



Wpisz nazwę projektu np. *project-jfx-client* i naciśnij przycisk *Finish*.



2.2. Jeżeli w katalogach projektu *src\main\java* lub *src\test\java* zostały utworzone automatycznie jakieś pliki to należy je teraz usunąć. Edytuj plik *build.gradle* i wpisz:

```
plugins {
 id 'java'
}

group = 'com.project'
version = '1.0'

repositories {
    jcenter()
}

dependencies {
      compile group: 'org.hsqldb', name: 'hsqldb', version: '2.5.0'
      compile group: 'org.hibernate', name: 'hibernate-core', version: '5.4.13.Final'
      compile group: 'ch.qos.logback', name: 'logback-classic', version: '1.2.3'
      compile group: 'ch.qos.logback', name: 'logback-core', version: '1.2.3'
      compile group: 'org.slf4j', name: 'slf4j-api', version: '1.7.30'
}
```

2.3. W katalogu *src\main\resources\META-INF* (utwórz nieistniejące podkatalogi) umieścić tzw. deskryptor utrwalania – plik konfiguracyjny *persistence.xml* (szczegółowy opis struktury pliku można znaleźć na stronie z oficjalną dokumentacją frameworku Hibernate – https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html).

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
   <persistence-unit name="hsqlManager" transaction-type="RESOURCE_LOCAL">
      <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
         <properties>
            <property name="javax.persistence.jdbc.driver" value="org.hsqldb.jdbcDriver" />
            <property name="javax.persistence.jdbc.user" value="admin" />
            <property name="javax.persistence.jdbc.password" value="admin" />
            <property name="javax.persistence.jdbc.url" value="jdbc:hsqldb:file:db/projekty" />
            <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect" />
            <property name="hibernate.hbm2ddl.auto" value="update" />
            <property name="hibernate.show_sql" value="true" />
      </properties>
   </persistence-unit>
</persistence>
```

Wartości *create*, *create-drop* lub *update* parametru *hibernate.hbm2ddl.auto* służą do automatycznego tworzenia lub aktualizacji struktury bazy danych na podstawie utworzonych klas encyjnych. Aplikacje wdrożone u klienta nie powinny używać tych wartości, poza tym drukowanie zapytań SQL (*hibernate.show_sql*) należy również wyłączać.

2.4. Kliknij prawym przyciskiem myszki na głównej ikonce projektu i z menu wybierz *Gradle -> Refresh Gradle Project*.

3. PODŁĄCZENIE MECHANIZMU REJESTRACJI

3.1. W podkatalogu *src\main\resources* stwórz plik *logback.xml* z przedstawioną poniżej zawartością. Kliknij prawym przyciskiem myszki na głównej ikonce projektu i wybierz *Gradle -> Refresh Gradle Project*.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="true">
   <property name="LOG_FILE" value="project-jfx-client" />
   <property name="LOG_DIR" value="logs" />
   <property name="LOG_ARCHIVE" value="${LOG_DIR}/archive" />

   <!-- Send messages to System.out -->
   <appender name="STDOUT"
      class="ch.qos.logback.core.ConsoleAppender">
      <encoder>
         <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36}.%M\(%line\) - %msg%n</pattern>
      </encoder>
   </appender>

   <!-- Save messages to a file -->
   <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
      <file>${LOG_DIR}/${LOG_FILE}.log</file>
      <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
         <!-- daily rollover -->
         <fileNamePattern>${LOG_ARCHIVE}/%d{yyyy-MM-dd}${LOG_FILE}.log.zip
         </fileNamePattern>
         <!-- keep 30 days' worth of history capped at 30MB total size -->
         <maxHistory>30</maxHistory>
         <totalSizeCap>30MB</totalSizeCap>
      </rollingPolicy>
      <encoder>
         <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36}.%M\(%line\) - %msg%n</pattern>
      </encoder>
   </appender>
```

```
   <!-- For the 'com.project' package and all its subpackages -->
   <logger name="com.project" level="INFO" additivity="false">
      <appender-ref ref="STDOUT" />
      <appender-ref ref="FILE" />
   </logger>

   <!-- By default, the level of the root level is set to INFO -->
   <root level="INFO">
      <appender-ref ref="STDOUT" />
   </root>
</configuration>
```

3.2. Zamiast korzystać z *System.out.println(...);* można teraz używać mechanizmu rejestracji, który oprócz standardowego drukowania komunikatów w konsoli będzie zapisywał również ich zawartość w plikach podkatalogu *logs*, a także automatycznie je archiwizował. Pamiętaj, że we wszystkich klasach, które mają korzystać z mechanizmu rejestracji trzeba tworzyć zmienną za pomocą statycznej metody *LoggerFactory.getLogger* przekazując w jej parametrze odpowiednią klasę. Poniżej przedstawione zostały przykłady prezentujące korzystanie z mechanizmu rejestracji.

```
package ...
...

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
...
public class JakasKlasa {
   private static final Logger logger = LoggerFactory.getLogger(JakasKlasa.class);
...

logger.info("Uruchamianie programu ...");
...
logger.info("Wersja aplikacji: {}", 1.9);

logger.warn("Uaktualnij aplikację. Najnowsza dostępna wersja: {}", 2.0);

...
} catch (SQLException e) {
   logger.error("Błąd podczas zapisywania projektu!", e);
   ...
   int kodBledu = 7;
   logger.error("Błąd podczas zapisywania projektu (kod błędu: {})!", kodBledu, e);
}
```

4. IMPLEMENTACJA APLIKACJI

4.1. Utwórz pakiet *com.project.model* i zdefiniuj w nim klasę, która będzie odwzorowaniem tabeli *projekt* (patrz rys. 4.1). W klasie wystarczy zdefiniować zmienne i dodać odpowiednie adnotacje JPA, natomiast konstruktory oraz tzw. akcesory można wygenerować automatycznie.

```
package com.project.model;

import java.time.LocalDate;
import java.time.LocalDateTime;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import org.hibernate.annotations.CreationTimestamp;

@Entity
@Table(name="projekt") //potrzebne tylko jeżeli nazwa tabeli w bazie danych ma być inna od nazwy klasy
public class Projekt {

	@Id
	@GeneratedValue
	@Column(name="projekt_id") //tylko jeżeli nazwa kolumny w bazie danych ma być inna od nazwy zmiennej
	private Integer projektId;
```

```
        @Column(nullable = false, length = 50)
        private String nazwa;

        @Column(length = 1000)
        private String opis;

        @CreationTimestamp
        @Column(name = "dataczas_utworzenia", nullable = false, updatable=false)
        private LocalDateTime dataCzasUtworzenia;

        @Column(name = "data_oddania")
        private LocalDate dataOddania;

    /* TODO Wygeneruj dla powyższych zmiennych akcesory (Source -> Generate Getters and Setters).
     * Dodaj również bezparametrowy konstruktor oraz drugi konstruktor uwzględniający wszystkie powyższe
     * zmienne, a także trzeci pomijający pola projektId oraz dataCzasUtworzenia.
     */

}
```

W klasach modelu można też korzystać z adnotacji spoza pakietu *javax.persistence* np. *@CreationTimestamp* i *@UpdateTimestamp*, które pozwalają na automatyczne przypisywanie dat i czasu podczas tworzenia lub modyfikacji rekordu np.:

```
import org.hibernate.annotations.CreationTimestamp;

// ...
@CreationTimestamp
@Column(name = "dataczas_utworzenia", nullable = false, updatable = false)
private LocalDateTime dataCzasUtworzenia;


import org.hibernate.annotations.UpdateTimestamp;

// ...
@UpdateTimestamp
@Column(name = "dataczas_modyfikacji", nullable = false)
private LocalDateTime dataCzasModyfikacji;
```

4.2. W pakiecie *com.project.model* utwórz klasę *Zadanie* odwzorowującą bazodanową tabelę *zadanie* (rys. 5.1).

```
package com.project.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="zadanie")
public class Zadanie {
        @Id
        @GeneratedValue
        @Column(name="zadanie_id")
        private Integer zadanieId;

        /*TODO Uzupełnij kod o zmienne reprezentujące pozostałe pola tabeli zadanie (patrz rys. 4.1),
        .       następnie wygeneruj dla nich akcesory (Source -> Generate Getters and Setters),
        .       ponadto dodaj pusty konstruktor oraz konstruktor ze zmiennymi nazwa, opis i kolejnosc.
        */

}
```

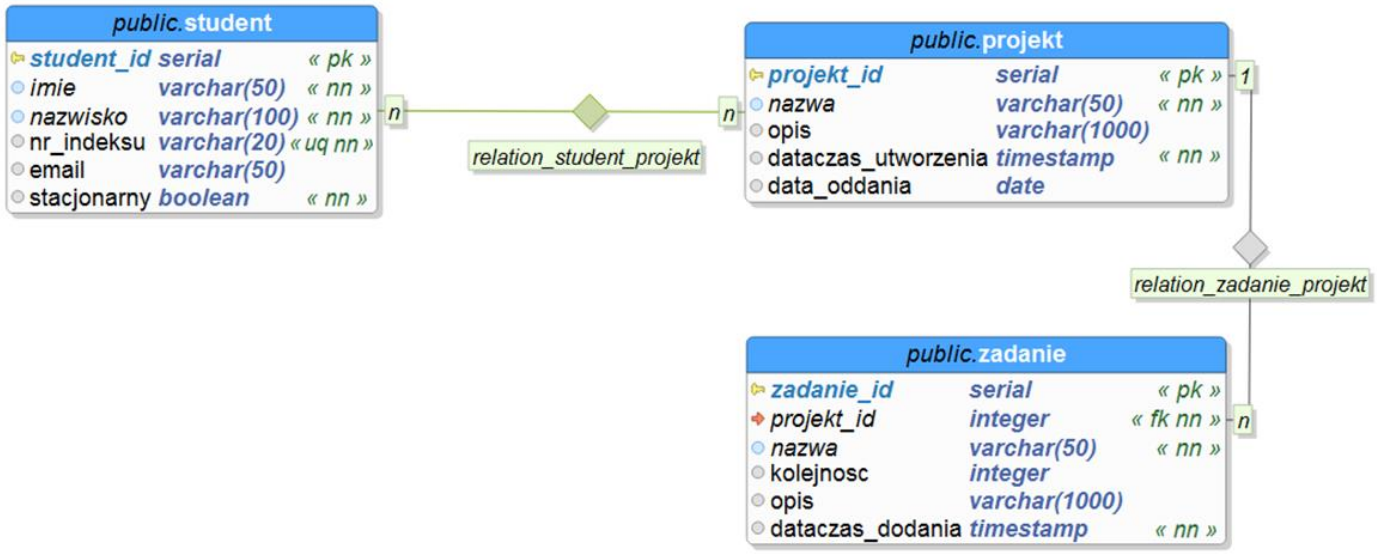
4.3. Realizacja dwukierunkowej relacji jeden do wielu. W klasie *Zadanie* dodaj zmienną *projekt* oraz adnotację *@ManyToOne*. Możesz też użyć adnotacji *@JoinColumn*. Wygeneruj akcesory dla nowo utworzonej zmiennej.

```
        @ManyToOne
        @JoinColumn(name = "projekt_id")
        private Projekt projekt;
```

W klasie *Projekt* dodaj listę *zadania* z adnotacją *@OneToMany* i parametrem *mappedBy*, którego wartość wskazuje zmienną po drugiej stronie relacji tj. *projekt* z klasy *Zadanie*. Pamiętaj o wygenerowaniu akcesorów.

```
        @OneToMany(mappedBy = "projekt")
        private List<Zadanie> zadania;
```

Rys. 4.1. Model bazy danych *projekty*



4.4. W pakiecie *com.project.model* utwórz klasę (z dodatkowymi konstruktorami) odwzorowującą tabelę *student* oraz zaimplementuj relację wiele do wielu między tabelami *projekt* i *student*.

```
@Entity
@Table(name = "student")
public class Student {

	/* TODO Uzupełnij kod o zmienne reprezentujące pola tabeli student (patrz rys. 4.1),
	.	następnie wygeneruj dla nich akcesory (Source -> Generate Getters and Setters)
	.
	*/

	public Student() {
	}

	public Student(String imie, String nazwisko, String nrIndeksu, Boolean stacjonarny) {
		this.imie = imie;
		this.nazwisko = nazwisko;
		this.nrIndeksu = nrIndeksu;
	}

	public Student(String imie, String nazwisko, String nrIndeksu, String email, Boolean stacjonarny) {
		this.imie = imie;
		this.nazwisko = nazwisko;
		this.nrIndeksu = nrIndeksu;
		this.email = email;
		this.stacjonarny = stacjonarny;
	}

	//...
}
```

Do nowo utworzonej klasy *Student* dodaj zmienną *projekty* z adnotacją *@ManyToMany* (*mappedBy* wskazuje na zmienną w klasie *Projekt*).

```
	@ManyToMany(mappedBy = "studenci")
	private Set<Projekt> projekty;
```

Do istniejącej klasy *Projekt* dodaj zmienną *studenci* z adnotacją *@ManyToMany* i w każdej z tych klas wygeneruj akcesory dla utworzonych zmiennych.

```
@ManyToMany
@JoinTable(name = "projekt_student",
   joinColumns = {@JoinColumn(name="projekt_id")},
   inverseJoinColumns = {@JoinColumn(name="student_id")})
private Set<Student> studenci;
```

4.5. Utworzenie singletonu z obiektem klasy *EntityManagerFactory*.

Aby zapisać nowy projekt w bazie danych można np. skorzystać z poniższego fragmentu kodu. Jednak w naszej aplikacji musimy zagwarantować, że obiekt *entityManagerFactory* będzie tworzony tylko raz po uruchomieniu aplikacji.

```
EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("hsqlManager");
EntityManager entityManager = entityManagerFactory.createEntityManager();

entityManager.getTransaction().begin(); // Tworzenie i modyfikacje obligatoryjnie w bloku transakcyjnym

Projekt projekt = new Projekt();
projekt.setNazwa("Projekt testowy");
// Ustawienie pozostałych danych projektu
   ...

entityManager.persist(projekt);

entityManager.getTransaction().commit();

entityManager.close();
entityManagerFactory.close();
```

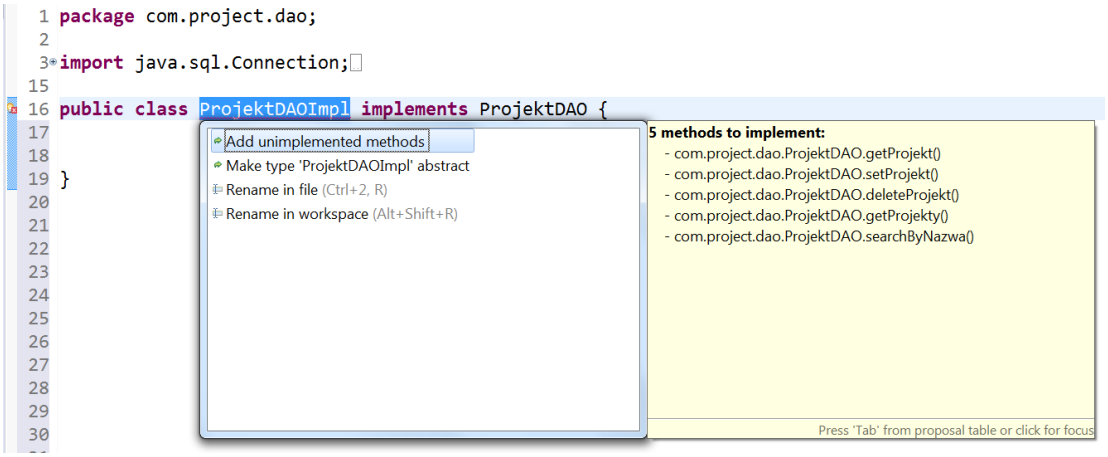
W tym celu możemy w pakiecie *com.project.datasource* utworzyć nową klasę – singleton *JPAUtil*.

```
package com.project.datasource;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class JPAUtil {
	private static final String DATA_SOURCE = "hsqlManager";//nazwa persistence-unit z pliku persistence.xml
	private static final EntityManagerFactory entityManagerFactory;
	static {
		entityManagerFactory = Persistence.createEntityManagerFactory(DATA_SOURCE);
	}

	private JPAUtil() {}

	public static EntityManager getEntityManager() {
		return entityManagerFactory.createEntityManager();
	}

	public static void close() {
		entityManagerFactory.close();
	}
}
```

4.6. Utwórz pakiet *com.project.dao* i zdefiniuj w nim interfejs *ProjektDAO*, który będzie zawierał definicje wszystkich metod pobierających i modyfikujących dane projektów przechowywanych w bazie. Zadaniem interfejsu jest hermetyzacja konkretnych implementacji. Dzięki temu, gdy dokonane zostaną jakiekolwiek zmiany w sposobie pozyskiwania danych, pozostała część aplikacji nie będzie wymagać modyfikacji.

```
package com.project.dao;

import java.util.List;
import com.project.model.Projekt;

public interface ProjektDAO {

	Projekt getProjekt(Integer projektId);

	void setProjekt(Projekt projekt);

	void deleteProjekt(Integer projektId);

	List<Projekt> getProjekty(Integer offset, Integer limit);

	List<Projekt> searchByNazwa(String search4, Integer offset, Integer limit);

}
```

4.7. Dodaj klasę *ProjektDAOImpl*, następnie po jej nazwie dopisz *implements ProjektDAO*. Aby dodać szkielety metod wymagających implementacji kliknij na znacznik błędu lewym przyciskiem myszki lub najedź kursorem na podkreślony fragment i wybierz *Add unimplemented methods*.



Zaimplementuj metody wykorzystując mapowanie obiektowo-relacyjne.

```
package com.project.dao;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import javax.persistence.TypedQuery;
import com.project.datasource.JPAUtil;
import com.project.model.Projekt;

public class ProjektDAOImpl implements ProjektDAO {

	@Override
	public Projekt getProjekt(Integer projektId) {
	   EntityManager entityManager = JPAUtil.getEntityManager();
	   Projekt projekt = entityManager.find(Projekt.class, projektId);
	   entityManager.close();
	   return projekt;
	}

	@Override
	public void setProjekt(Projekt projekt) {
	   // TODO
	}

	@Override
	public void deleteProjekt(Integer projektId) {
	   // TODO
	}

	@Override
	public List<Projekt> getProjekty(Integer offset, Integer limit) {
	   // TODO
	   return null;
	}

	@Override
	public List<Projekt> searchByNazwa(String search4, Integer offset, Integer limit) {
	   // TODO
	   return null;
	}
}
```

Przykłady operacji bazodanowych:

– usuwanie projektu z bazy danych (wykonywane w bloku transakcyjnym
tj. pomiędzy *entityManager.getTransaction().begin()* i *entityManager.getTransaction().commit()*)

```
entityManager.remove(projekt);
```

– wyszukiwanie projektu za pomocą identyfikatora przekazywanego w drugim parametrze metody *find*.

```
int projektId = 1;
Projekt projekt = entityManager.find(Projekt.class, projektId);
```

– wyszukiwanie za pomocą zapytań w JPQL. Uwaga! W zapytaniu używamy zawsze nazw klasy i jej pól a nie rzeczywistych nazw pól i tabel w bazie danych!

```
TypedQuery<Projekt> query = entityManager
    .createQuery("SELECT p FROM Projekt p WHERE p.projektId = 1", Projekt.class);
Projekt projekt = query.getSingleResult();
```

– pobieranie listy wyników za pomocą JPQL

```
TypedQuery<Projekt> query = entityManager
    .createQuery("SELECT p FROM Projekt p WHERE p.projektId > 1", Projekt.class);
List<Projekt> projekty = query.getResultList();
```

– pobieranie wyników za pomocą sparametryzowanego zapytania w JPQL

```
TypedQuery<Projekt> query = entityManager
    .createQuery("SELECT p FROM Projekt p WHERE p.projektId > :id", Projekt.class);
query.setParameter("id", 1);
List<Projekt> projekty = query.getResultList();
```

– pobieranie wyników za pomocą sparametryzowanego zapytania w JPQL (z użyciem listy)

```
TypedQuery<Projekt> query = entityManager
          .createQuery("SELECT p FROM Projekt p WHERE p.projektId IN :ids", Projekt.class);
List<Integer> ids = new ArrayList<Integer>();
ids.add(1);
ids.add(2);
query.setParameter("ids", ids);

List<Projekt> projekty = query.getResultList();
```

– utrwalanie nowych obiektów (wykonywane w bloku transakcyjnym)

```
entityManager.persist(projekt);
```

– modyfikacja obiektów (wykonywana w bloku transakcyjnym)

```
projektKopia = entityManager.merge(projekt); // zwracany obiekt projektKopia jest encją
                                             // zarządzaną, a projekt encją odłączoną
```

– odświeżenie stanu zarządzanej encji danymi z bazy

```
entityManager.refresh(projekt);
```

– stronicowanie

```
TypedQuery<Projekt> query = entityManager.createQuery("SELECT p FROM Projekt p ORDER BY
                                                       p.dataCzasUtworzenia DESC", Projekt.class);
query.setFirstResult(offset);
query.setMaxResults(limit);
List<Projekt> projekty = query.getResultList();
```

– parametr *fetch*

Wartość parametru *fetch* równa *FetchType.LAZY* (jest to wartość domyślna, nie wymaga jawnego deklarowania) oznacza, że powiązane dane są pobierane z bazy dopiero w momencie ich pierwszego użycia. Nawet jeżeli pobierzemy dane np. za pomocą poniższego kodu

```
int projektId = 1;
Projekt projekt = entityManager.find(Projekt.class, projektId);
```

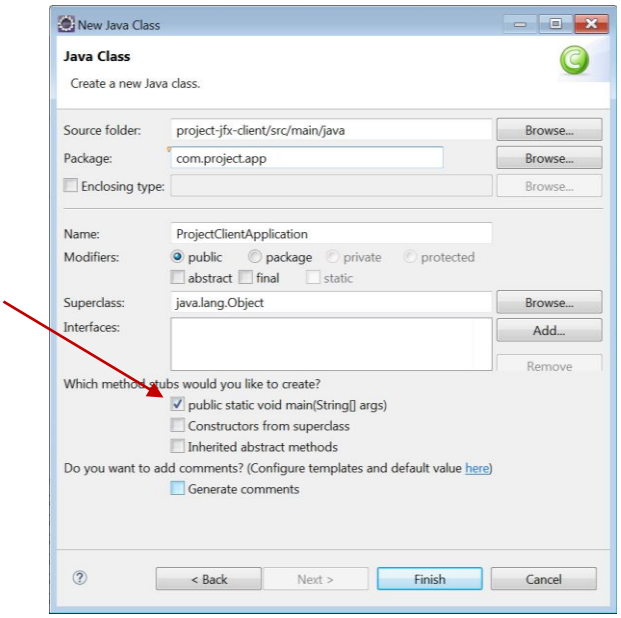
to realizująca relację jeden do wielu lista *zadania* klasy *Projekt* będzie w rzeczywistości pusta. Dopiero pierwsze wywołanie metody *projekt.getZadania()* pobierającej listę encji powoduje chwilę przed jej zwróceniem wykonanie bazodanowego zapytania ustawiającego dane listy. Jest to bardzo użyteczny mechanizm zwiększający wydajność, jeżeli jednak zachodzi potrzeba wymuszenia w naszym programie natychmiastowego pobierania powiązanych danych to musimy użyć *FetchType.EAGER*, np.

```
@OneToMany(fetch = FetchType.EAGER)
private List<Zadanie> zadania;
```

– dokonywanie masowych zmian w bazie

```
int alteredRows = em.createQuery("UPDATE Student s SET s.stacjonarny = true").executeUpdate();
```

4.8. W pakiecie *com.project.app* utwórz klasę *ProjectClientApplication* zawierającą metodę *main*.



Następnie sprawdź poprawność metody zapisującej nowy projekt oraz pobierającej dane projektu dla wskazanego identyfikatora. Aby uruchomić aplikację kliknij prawym przyciskiem myszki wewnątrz okna z kodem źródłowym klasy *ProjectClientApplication* i wybierz *Run As -> Java Application*.

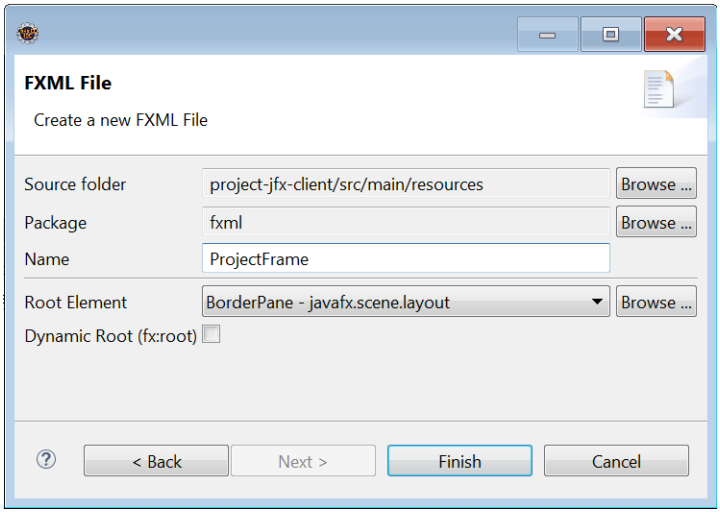
```
package com.project.app;

import java.time.LocalDate;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.project.dao.ProjektDAO;
import com.project.dao.ProjektDAOImpl;
import com.project.model.Projekt;

public class ProjectClientApplication {

   private static final Logger logger = LoggerFactory.getLogger(ProjectClientApplication.class);

   public static void main(String[] args) {

       ProjektDAO projektDAO = new ProjektDAOImpl();

        ...

       try {
          Projekt projekt = new Projekt("Projekt testowy", "Opis testowy", LocalDate.of(2020,  06,  22));
          projektDAO.setProjekt(projekt);
          logger.info("Id utworzonego projektu: {}", projekt.getProjektId());
          //System.out.println("Id utworzonego projektu: " + projekt.getProjektId());
       } catch (RuntimeException e) {
          logger.error("Błąd podczas dodawania projektu!", e);
       }

       //TODO Pobieranie danych

   }
}
```

5. GRAFICZNY INTERFEJS UŻYTKOWNIKA

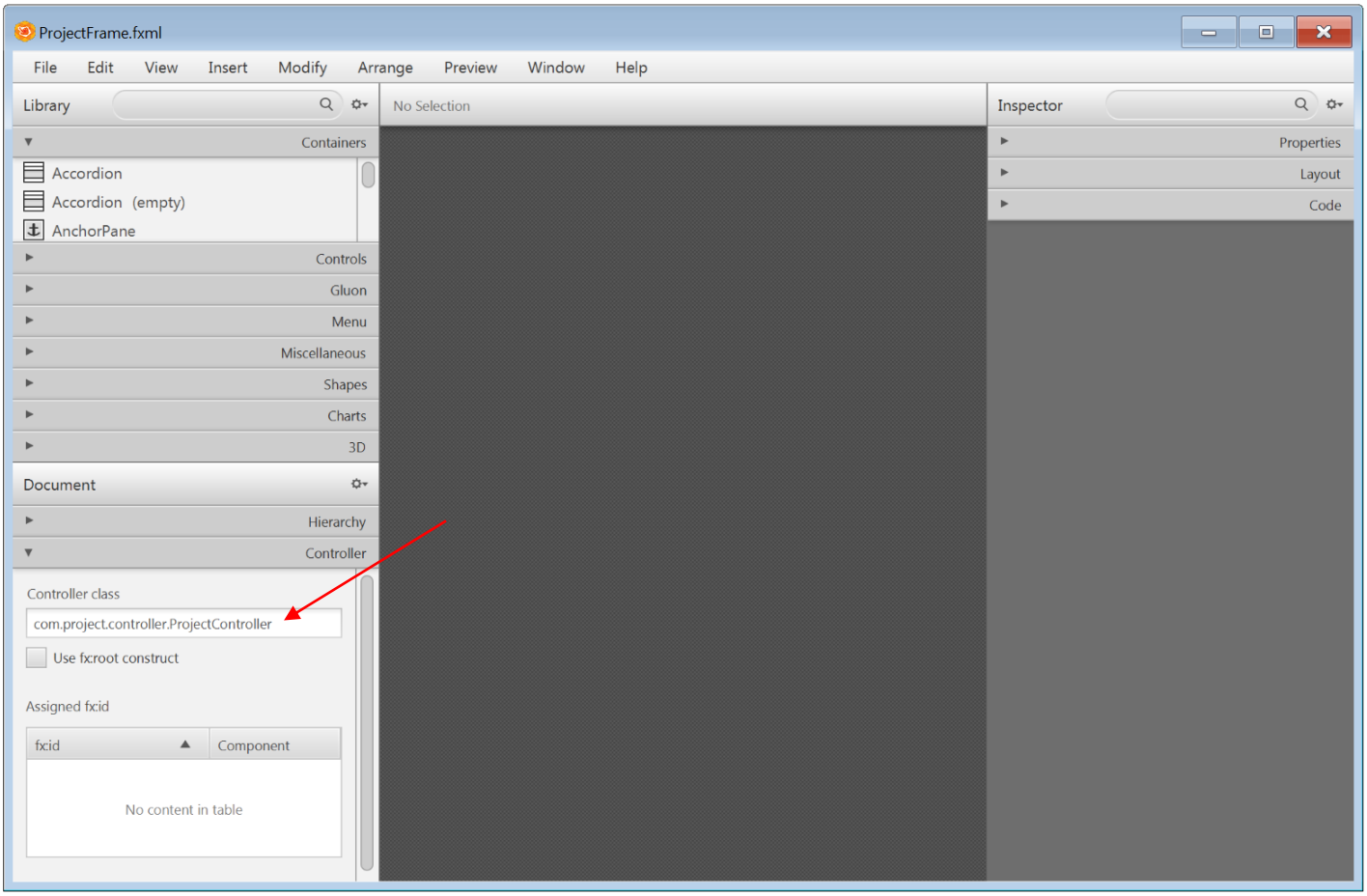
5.1. W katalogu *src\main\resources* utwórz podkatalog *css*, następnie dodaj do niego plik *application.css* z przedstawioną poniżej zawartością.

```
/* JavaFX CSS - Leave this comment until you have at least create one rule which uses -fx-Property */
.table-column {
  -fx-alignment: CENTER_LEFT;
}

.table-row-cell {
    -fx-background-color: #F5F5F5;
    -fx-background-insets: 0.0, 0.0 0.0 1.0 0.0;
    -fx-text-fill: black;
    -fx-border-width: 1.0 1.0 0.0 0.0;
    -fx-border-color: #DBD7D2;
    -fx-table-cell-border-color: transparent;
}

.table-row-cell:odd {
       -fx-background-color: #EAE9E8;
}

.table-view:focused .table-row-cell:filled:focused:selected {
    -fx-background-color: -fx-focus-color, -fx-cell-focus-inner-border, -fx-selection-bar;
    -fx-background-insets: 0.0, 1.0, 2.0;
    -fx-background: -fx-accent;
    -fx-text-fill: -fx-selection-bar-text;
}
```

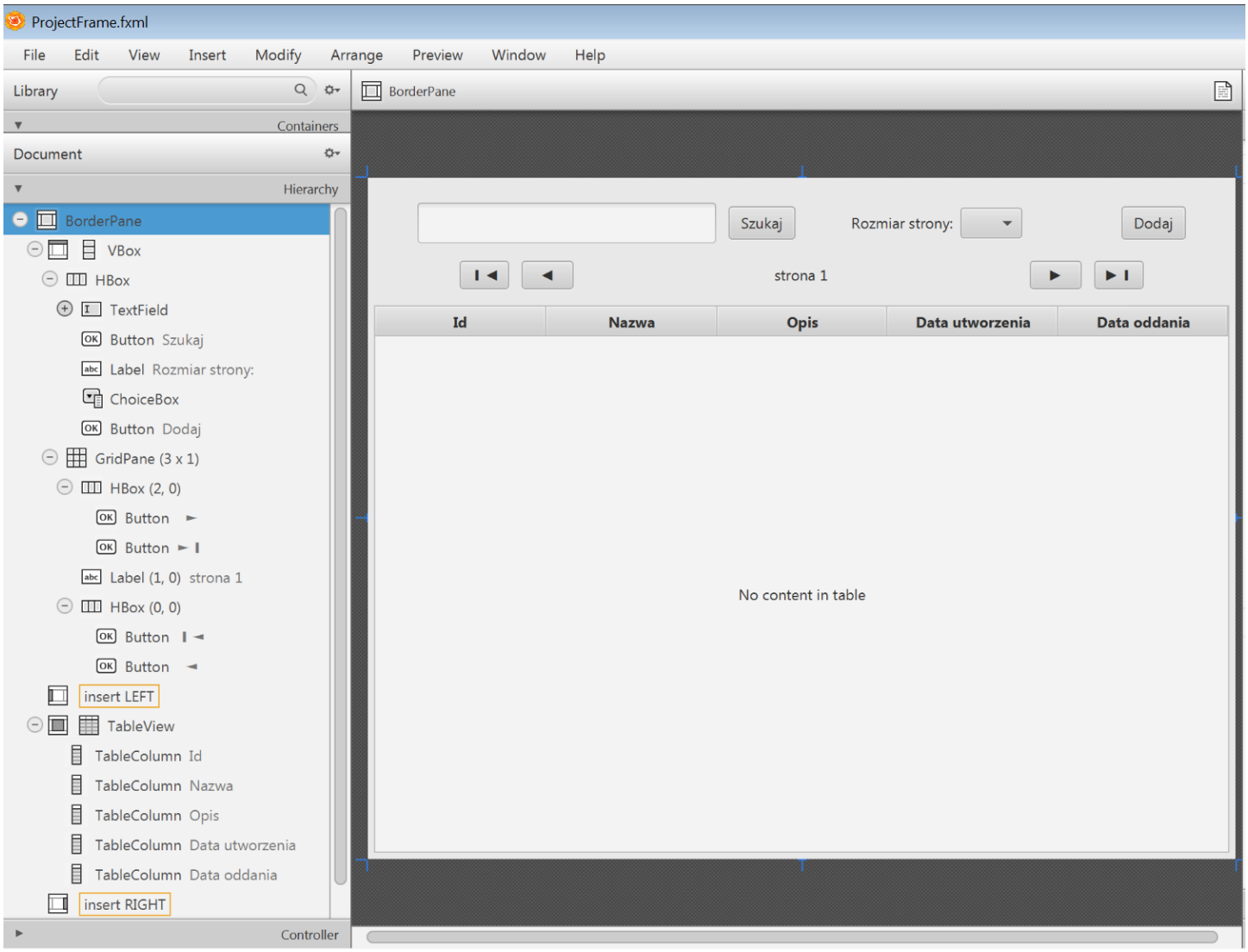
5.2. W katalogu *src\main\resources* utwórz podkatalog *fxml*. Wybierz z menu *File -> New -> Other* (lub użyj skrótu *CTRL + N*), następnie zaznacz *New FXML Document* (z *JavaFX*) i kliknij *Next*. W nowo otwartym oknie wpisz nazwę pliku *ProjectFrame*, a także z listy *Root Element* wybierz *BorderPane* i naciśnij *Finish*. Ponadto w pakiecie *com.project.controller* utwórz klasę kontrolera o nazwie *ProjectController*.



5.3. Kliknij prawym przyciskiem myszki na pliku *src\main\resources\fxml\ProjectFrame.fxml* i wybierz *Open with SceneBuilder*. W polu *Controller class* wpisz klasę kontrolera poprzedzoną nazwą pakietu.



Następnie zaprojektuj przedstawione poniżej okno aplikacji. Aby utworzyć tekst przycisków nawigacji można użyć znaków *Unicode* (najlepiej wpisać je bezpośrednio w pliku *ProjectFrame.fxml*) np. ⏮ (\u2759\u25C4), ◀ (\u25C4), ▶ (\u25BA), ⏭ (\u25BA\u2759).
Pamiętaj, aby przed zamknięciem SceneBuildera zapisać zmiany!

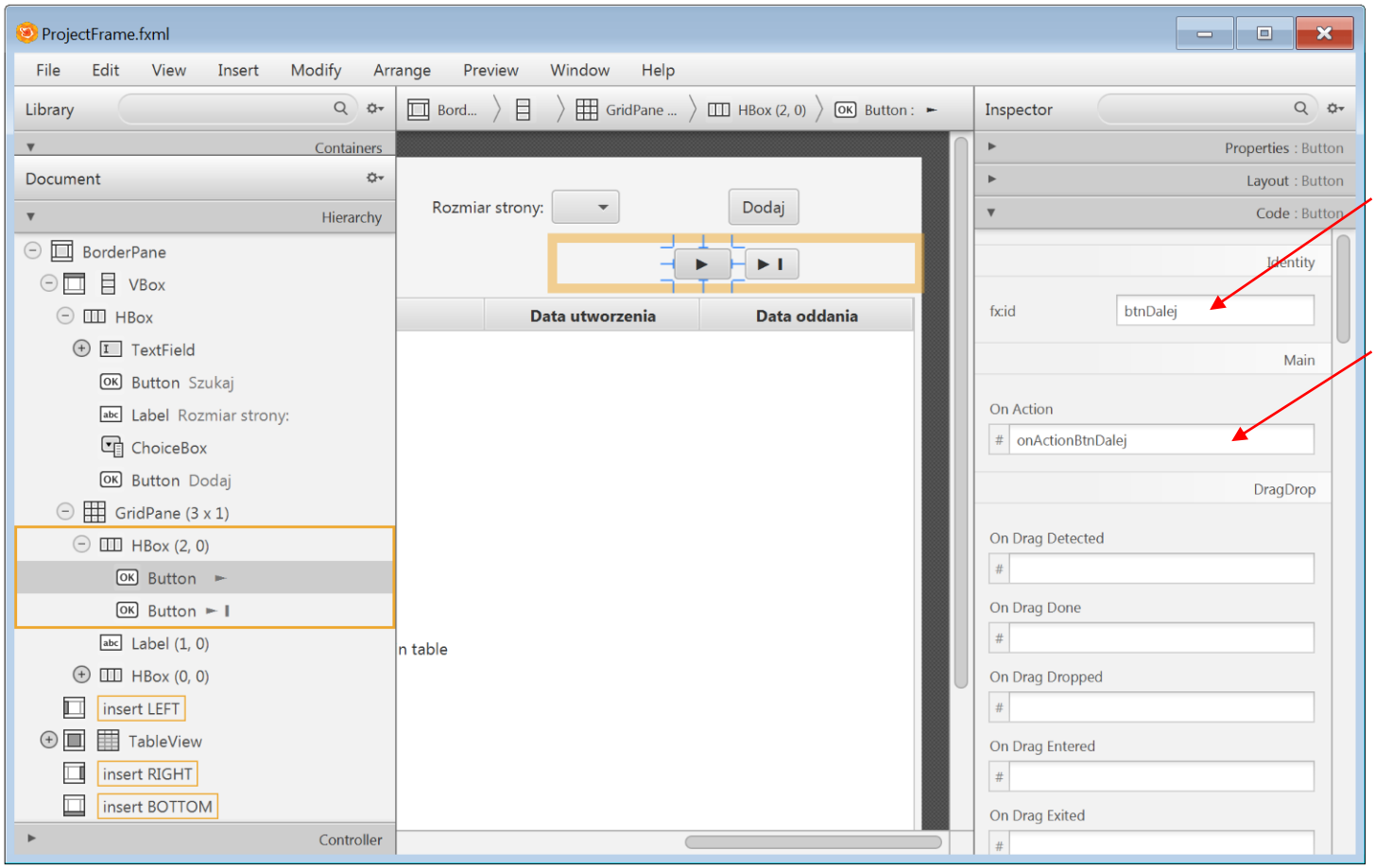


5.4. W klasie *ProjectController* trzeba dodać m.in. zmienne komponentów, do których dzięki adnotacji *@FXML* JavaFX wstrzyknie utworzone przez nią obiekty interfejsu użytkownika. Poza tym należy również zdefiniować grupę metod oznaczonych tą samą adnotacją, których celem będzie realizacja zadań wykonywanych po wciśnięciu przycisków, a także automatycznie wywoływaną podczas tworzenia kontrolera metodę *initialize*.

```
package com.project.controller;

import java.time.LocalDate;
import java.time.LocalDateTime;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.project.model.Projekt;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;

public class ProjectController {
   private static final Logger logger = LoggerFactory.getLogger(ProjectController.class);

   //Zmienne do obsługi stronicowania i wyszukiwania
   private String search4;
   private Integer pageNo;
   private Integer pageSize;

   //Automatycznie wstrzykiwane komponenty GUI
   @FXML
   private ChoiceBox<Integer> cbPageSizes;
   @FXML
   private TableView<Projekt> tblProjekt;
```

```
    @FXML
    private TableColumn<Projekt, Integer> colId;
    @FXML
    private TableColumn<Projekt, String> colNazwa;
    @FXML
    private TableColumn<Projekt, String> colOpis;
    @FXML
    private TableColumn<Projekt, LocalDateTime> colDataCzasUtworzenia;
    @FXML
    private TableColumn<Projekt, LocalDate> colDataOddania;
    @FXML
    private TextField txtSzukaj;

    public ProjectController() { //Utworzeniu pustego konstruktora jest obligatoryjne!
    }

    @FXML
    public void initialize() { //Metoda automatycznie wywoływana przez JavaFX zaraz po wstrzyknięciu
        search4 = null;         //wszystkich komponentów. Uwaga! Wszelkie modyfikacje komponentów
        pageNo = 0;             //(np. cbPageSizes) trzeba realizować wewnątrz tej metody. Nigdy
        pageSize = 10;          //nie używaj do tego celu konstruktora.

        cbPageSizes.getItems().addAll(5, 10, 20, 50, 100);
        cbPageSizes.setValue(pageSize);
    }

    //Grupa metod do obsługi przycisków
    @FXML
    private void onActionBtnSzukaj(ActionEvent event) {
    }

    @FXML
    private void onActionBtnDalej(ActionEvent event) {
    }

    @FXML
    private void onActionBtnWstecz(ActionEvent event) {
    }

    @FXML
    private void onActionBtnPierwsza(ActionEvent event) {
    }

    @FXML
    private void onActionBtnOstatnia(ActionEvent event) {
    }

    @FXML
    private void onActionBtnDodaj(ActionEvent event) {
    }
}
```

Uwaga! Nazwy zmiennych oraz metod obsługujących zdarzenia w klasie *ProjectController* muszą być identyczne jak te przypisane w atrybutach *fx:id* i *onAction* komponentów zdefiniowanych w pliku *src\main\resources\fxml\ProjectFrame.fxml*. Aby utworzyć atrybuty edytuj plik FXML za pomocą SceneBuildera, wybieraj kolejne elementy klikając na nich i wpisuj odpowiednie nazwy w zaznaczonych na poniższym rysunku polach tekstowych zakładki *Code*.



5.5. Zmodyfikuj utworzoną w p. 5.6 klasę *ProjectClientApplication* tak jak pokazano poniżej. Aby uruchomić aplikację kliknij prawym przyciskiem myszki wewnątrz okna z kodem źródłowym lub na ikonce klasy i wybierz *Run As -> Java Application.*

```
package com.project.app;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class ProjectClientApplication extends Application {
   private Parent root;
   private FXMLLoader fxmlLoader;

   public static void main(String[] args) {
      Launch(ProjectClientApplication.class, args);
   }

   @Override
   public void start(Stage primaryStage) throws Exception {
      fxmlLoader = new FXMLLoader();
      fxmlLoader.setLocation(getClass().getResource("/fxml/ProjectFrame.fxml"));
      root = fxmlLoader.load();
      primaryStage.setTitle("Projekty");
      Scene scene = new Scene(root);
      scene.getStylesheets().add(getClass().getResource("/css/application.css").toExternalForm());
      primaryStage.setScene(scene);
      primaryStage.sizeToScene();
      primaryStage.show();
   }

}
```

5.6. Inicjalizacja kolumn tabeli oraz utworzenie w kontrolerze listy typu *ObservableList*, w której będą przechowywane pobrane z bazy dane. Zaletą użycia *ObservableCollections* z JavaFX jest fakt, że gdy zmodyfikujemy taką kolekcję to element GUI z nią powiązany zostanie automatycznie zaktualizowany.

```
package com.project.controller;
 ...
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.collections.ObservableList;
import javafx.collections.FXCollections;

public class ProjectController {
    ...
   private ObservableList<Projekt> projekty;

    ...
   @FXML
   public void initialize() {
       ...
      colId.setCellValueFactory(new PropertyValueFactory<Projekt, Integer>("projektId"));
      colNazwa.setCellValueFactory(new PropertyValueFactory<Projekt, String>("nazwa"));
      colOpis.setCellValueFactory(new PropertyValueFactory<Projekt, String>("opis"));
      colDataCzasUtworzenia.setCellValueFactory(new PropertyValueFactory<Projekt, LocalDateTime>
                                                                          ("dataCzasUtworzenia"));
      colDataOddania.setCellValueFactory(new PropertyValueFactory<Projekt, LocalDate>("dataOddania"));

      projekty = FXCollections.observableArrayList();
      //Powiązanie tabeli z listą typu ObservableList przechowującą projekty
      tblProjekt.setItems(projekty);
   }

   ...
}
```

5.7. Dodaj pulę wątków – obiekt klasy *ExecutorService*, który zarządza tworzeniem nowych oraz wykonuje „recykling" zakończonych wątków. Ponadto utwórz obiekt *DAO* zapewniający dostęp do danych oraz metodę *loadPage*, której zadaniem będzie pobieranie w nowym wątku wskazanego podzbioru projektów. Trzeba również pamiętać, że przed zamknięciem aplikacji z obiektu klasy *ExecutorService* powinna być wywoływana jedna z metod - *shutdown* lub *shutdownNow*. W tym celu w kontrolerze dodajemy publiczną metodę o nazwie *shutdown*.

```
package com.project.controller;
 ...
import javafx.application.Platform;
import com.project.dao.ProjektDAO;
import com.project.dao.ProjektDAOImpl;
import com.project.model.Projekt;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ProjectController {
    ...
   private ExecutorService wykonawca;
   private ProjektDAO projektDAO;

    ...
   @FXML
   public void initialize() {
       ...
      projektDAO = new ProjektDAOImpl();
      wykonawca = Executors.newFixedThreadPool(1); // W naszej aplikacji wystarczy jeden wątek do pobierania
                                                   // danych. Przekazanie większej ilości takich zadań do
      loadPage(search4, pageNo, pageSize);         // puli jednowątkowej powoduje ich kolejkowanie i sukcesywne
   }                                               // wykonywanie.
```

```
private void loadPage(String search4, Integer pageNo, Integer pageSize) {
   wykonawca.execute(() -> {
      try {
         List<Projekt> projektList = search4 == null || search4.isEmpty()
                  ? projektDAO.getProjekty(pageNo * pageSize, pageSize)
                  : projektDAO.searchByNazwa(search4, pageNo * pageSize, pageSize);
         if(projektList != null) {
            Platform.runLater(() -> {
               projekty.clear();
               projekty.addAll(projektList);
            });
          }
      } catch (RuntimeException e) {
         String errorInfo = "Błąd podczas pobierania listy projektów!";
         String errorDetails = e.getMessage();
         logger.error("{} ({})", errorInfo, e);
         Platform.runLater(() -> showError(errorInfo, errorDetails));
      }
   });
}

/** Metoda pomocnicza do prezentowania użytkownikowi informacji o błędach */
private void showError(String header, String content) {
   Alert alert = new Alert(AlertType.ERROR);
   alert.setTitle("Błąd");
   alert.setHeaderText(header);
   alert.setContentText(content);
   alert.showAndWait();
}

public void shutdown() {
  // Wystarczyłoby tylko samo wywołanie metody wykonawca.shutdownNow(), ale można również, tak jak poniżej,
  // zaimplementować wersję z oczekiwaniem na zakończenie wszystkich zadań wykonywanych w puli wątków.
  if(wykonawca != null) {
      wykonawca.shutdown();
      try {
         if(!wykonawca.awaitTermination(3, TimeUnit.SECONDS))
            wykonawca.shutdownNow();
      } catch (InterruptedException e) {
         wykonawca.shutdownNow();
      }
   }
}
```

5.8. Modyfikujemy metodę *start* klasy *ProjectClientApplication*, aby przy zamykaniu aplikacji była wywoływana metoda *shutdown* kontrolera.

```
package com.project.app;
 ...
import com.project.controller.ProjectController;
import javafx.application.Platform;

public class ProjectClientApplication extends Application {
    ...

  @Override
  public void start(Stage primaryStage) throws Exception {
     fxmlLoader = new FXMLLoader();
     fxmlLoader.setLocation(getClass().getResource("/fxml/ProjectFrame.fxml"));
     root = fxmlLoader.load();
     primaryStage.setTitle("Projekty");
     Scene scene = new Scene(root);
     scene.getStylesheets().add(getClass().getResource("/css/application.css").toExternalForm());

     ProjectController controller = fxmlLoader.getController();
     primaryStage.setOnCloseRequest(event -> {
       primaryStage.hide();
       controller.shutdown();
       Platform.exit();
       System.exit(0);
     });

     primaryStage.setScene(scene);
     primaryStage.sizeToScene();
     primaryStage.show();
  }
```

```
    ...
  }
```

5.9. Formatowanie daty i czasu.

```
package com.project.controller;
 ...
public class ProjectController {
    ...
  private static final DateTimeFormatter dateFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
  private static final DateTimeFormatter dateTimeFormater = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    ...
   @FXML
   public void initialize() {
       ...
      colDataCzasUtworzenia.setCellFactory(column -> new TableCell<Projekt, LocalDateTime>() {
         @Override
         protected void updateItem(LocalDateTime item, boolean empty) {
            super.updateItem(item, empty);
            if (item == null || empty) {
              setText(null);
            } else {
              setText(dateTimeFormater.format(item));
            }
         }
      });

   }
   ...
}
```

5.10. Tworzenie projektu. Do klasy kontrolera trzeba dodać przedstawioną poniżej metodę *edytujProjekt* oraz metodę pomocniczą *getRightLabel*. Następnie należy wstawić w *onActionBtnDodaj* wywołanie metody *edytujProjekt* przekazując w jej parametrze nowo utworzony, pusty projekt.

```
package com.project.controller;

...
import java.time.LocalDate;
import java.util.concurrent.ExecutorService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.project.dao.ProjektDAO;
import com.project.model.Projekt;
import java.util.Optional;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;
import javafx.fxml.FXML;
import javafx.application.Platform;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.ButtonBar.ButtonData;
import javafx.scene.control.Button;
import javafx.scene.control.ButtonType;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.control.TableView;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.util.Callback;
import javafx.util.StringConverter;
```

```
public class ProjectController {
     ...

     @FXML
     public void onActionBtnDodaj(ActionEvent event) {
          edytujProjekt(new Projekt());
     }

     ...

     private void edytujProjekt(Projekt projekt) {
          Dialog<Projekt> dialog = new Dialog<>();
          dialog.setTitle("Edycja");
          if (projekt.getProjektId() != null) {
               dialog.setHeaderText("Edycja danych projektu");
          } else {
               dialog.setHeaderText("Dodawanie projektu");
          }
          dialog.setResizable(true);
          Label lblId = getRightLabel("Id: ");
          Label lblNazwa = getRightLabel("Nazwa: ");
          Label lblOpis = getRightLabel("Opis: ");
          Label lblDataCzasUtworzenia = getRightLabel("Data utworzenia: ");
          Label lblDataOddania = getRightLabel("Data oddania: ");
          Label txtId = new Label();
          if (projekt.getProjektId() != null)
               txtId.setText(projekt.getProjektId().toString());
          TextField txtNazwa = new TextField();
          if (projekt.getNazwa() != null)
               txtNazwa.setText(projekt.getNazwa());
          TextArea txtOpis = new TextArea();
          txtOpis.setPrefRowCount(6);
          txtOpis.setPrefColumnCount(40);
          txtOpis.setWrapText(true);
          if (projekt.getOpis() != null)
               txtOpis.setText(projekt.getOpis());
          Label txtDataUtworzenia = new Label();
          if (projekt.getDataCzasUtworzenia() != null)
               txtDataUtworzenia.setText(dateTimeFormater.format(projekt.getDataCzasUtworzenia()));

          DatePicker dtDataOddania = new DatePicker();
          dtDataOddania.setPromptText("RRRR-MM-DD");
          dtDataOddania.setConverter(new StringConverter<LocalDate>() {
               @Override
               public String toString(LocalDate date) {
                    return date != null ? dateFormatter.format(date) : null;
               }

               @Override
               public LocalDate fromString(String text) {
                    return text == null || text.trim().isEmpty() ? null : LocalDate.parse(text, dateFormatter);
               }
          });
          dtDataOddania.getEditor().focusedProperty().addListener((obsValue, oldFocus, newFocus) -> {
               if (!newFocus) {
                    try {
                         dtDataOddania.setValue(dtDataOddania.getConverter().fromString(
                                                                 dtDataOddania.getEditor().getText()));
                    } catch (DateTimeParseException e) {
                         dtDataOddania.getEditor().setText(dtDataOddania.getConverter()
                                                                 .toString(dtDataOddania.getValue()));
                    }
               }
          });
          if (projekt.getDataOddania() != null) {
               dtDataOddania.setValue(projekt.getDataOddania());
          }

          GridPane grid = new GridPane();
          grid.setHgap(10);
          grid.setVgap(10);
          grid.setPadding(new Insets(5, 5, 5, 5));
          grid.add(lblId, 0, 0);
          grid.add(txtId, 1, 0);
          grid.add(lblDataCzasUtworzenia, 0, 1);
```

```
        grid.add(txtDataUtworzenia, 1, 1);
        grid.add(lblNazwa, 0, 2);
        grid.add(txtNazwa, 1, 2);
        grid.add(lblOpis, 0, 3);
        grid.add(txtOpis, 1, 3);
        grid.add(lblDataOddania, 0, 4);
        grid.add(dtDataOddania, 1, 4);
        dialog.getDialogPane().setContent(grid);

        ButtonType buttonTypeOk = new ButtonType("Zapisz", ButtonData.OK_DONE);
        ButtonType buttonTypeCancel = new ButtonType("Anuluj", ButtonData.CANCEL_CLOSE);
        dialog.getDialogPane().getButtonTypes().add(buttonTypeOk);
        dialog.getDialogPane().getButtonTypes().add(buttonTypeCancel);
        dialog.setResultConverter(new Callback<ButtonType, Projekt>() {
            @Override
            public Projekt call(ButtonType butonType) {
                if (butonType == buttonTypeOk) {
                    projekt.setNazwa(txtNazwa.getText().trim());
                    projekt.setOpis(txtOpis.getText().trim());
                    projekt.setDataOddania(dtDataOddania.getValue());
                    return projekt;
                }
                return null;
            }
        });

        Optional<Projekt> result = dialog.showAndWait();
        if (result.isPresent()) {
            wykonawca.execute(() -> {
                try {
                    projektDAO.setProjekt(projekt);
                    Platform.runLater(() -> {
                        if (tblProjekt.getItems().contains(projekt)) {
                            tblProjekt.refresh();
                        } else {
                            tblProjekt.getItems().add(0, projekt);
                        }
                    });
                } catch (RuntimeException e) {
                    String errorInfo = "Błąd podczas zapisywania danych projektu!";
                    String errorDetails = e.getMessage();
                    logger.error("{} ({})", errorInfo, e);
                    Platform.runLater(() -> showError(errorInfo, errorDetails));
                }
            });
        }
    }

    private Label getRightLabel(String text) {
        Label lbl = new Label(text);
        lbl.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
        lbl.setAlignment(Pos.CENTER_RIGHT);
        return lbl;
    }
```

5.11. Edycja projektu. W metodzie *initialize* klasy *ProjectController*, po fragmencie inicjalizującym kolumny tabeli, dodajemy nową kolumnę z przyciskami *Edytuj* i *Usuń*. Można również ustawić względną szerokość poszczególnych kolumn za pomocą *setMaxWidth*.

```
@FXML
public void initialize() {
  ...

    colId.setCellValueFactory(new PropertyValueFactory<Projekt, Integer>("projektId"));
    colNazwa.setCellValueFactory(new PropertyValueFactory<Projekt, String>("nazwa"));
    colOpis.setCellValueFactory(new PropertyValueFactory<Projekt, String>("opis"));
    colDataCzasUtworzenia.setCellValueFactory(new PropertyValueFactory<Projekt,
                                                        LocalDateTime>("dataCzasUtworzenia"));
    colDataOddania.setCellValueFactory(new PropertyValueFactory<Projekt, LocalDate>("dataOddania"));
```

```
    //Utworzenie nowej kolumny
    TableColumn<Projekt, Void> colEdit = new TableColumn<>("Edycja");
    colEdit.setCellFactory(column -> new TableCell<Projekt, Void>() {
        private final GridPane pane;
        { //Blok inicjalizujący w anonimowej klasie wewnętrznej
           Button btnEdit = new Button("Edycja");
           Button btnRemove = new Button("Usuń");
           btnEdit.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
           btnRemove.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
           btnEdit.setOnAction(event -> {
              edytujProjekt(getCurrentProjekt());
           });
           btnRemove.setOnAction(event -> {
             //TODO
             //usunProjekt(getCurrentProjekt());
           });
           pane = new GridPane();
           pane.setAlignment(Pos.CENTER);
           pane.setHgap(10);
           pane.setVgap(10);
           pane.setPadding(new Insets(5, 5, 5, 5));
           pane.add(btnEdit, 0, 0);
           pane.add(btnRemove, 0, 1);
        }

        private Projekt getCurrentProjekt() {
           int index = this.getTableRow().getIndex();
           return this.getTableView().getItems().get(index);
        }

        @Override
        protected void updateItem(Void item, boolean empty) {
           super.updateItem(item, empty);
           setGraphic(empty ? null : pane);
        }
    });

    //Dodanie kolumny do tabeli
    tblProjekt.getColumns().add(colEdit);

    //Ustawienie względnej szerokości poszczególnych kolumn
    colId.setMaxWidth(5000);
    colNazwa.setMaxWidth(10000);
    colOpis.setMaxWidth(10000);
    colDataCzasUtworzenia.setMaxWidth(9000);
    colDataOddania.setMaxWidth(7000);
    colEdit.setMaxWidth(7000);

    ...
}
```

5.12. Usuwanie projektu.
W klasie *ProjectController* zaimplementuj metodę `private void usunProjekt(Projekt projekt)`. Podczas usuwania powinno pojawiać się okno dialogowe żądające potwierdzenia operacji. Wywołanie metody musi znaleźć się we fragmencie (zawartym w *initialize*) rejestrującym odbiorcę zdarzeń dla przycisku *btnRemove* tj.

```
btnRemove.setOnAction(event -> {
   usunProjekt(getCurrentProjekt());
});
```

5.13. Stronicowanie i wyszukiwanie projektu.
Zaimplementuj wyszukiwanie projektu za pomocą słowa wpisywanego w polu tekstowym *txtSzukaj*. Mechanizm wyszukiwania powinien uwzględniać bazodanową kolumnę *nazwa*. Dodaj również obsługę przycisków w metodach *onActionBtnDalej* i *onActionBtnWstecz*.

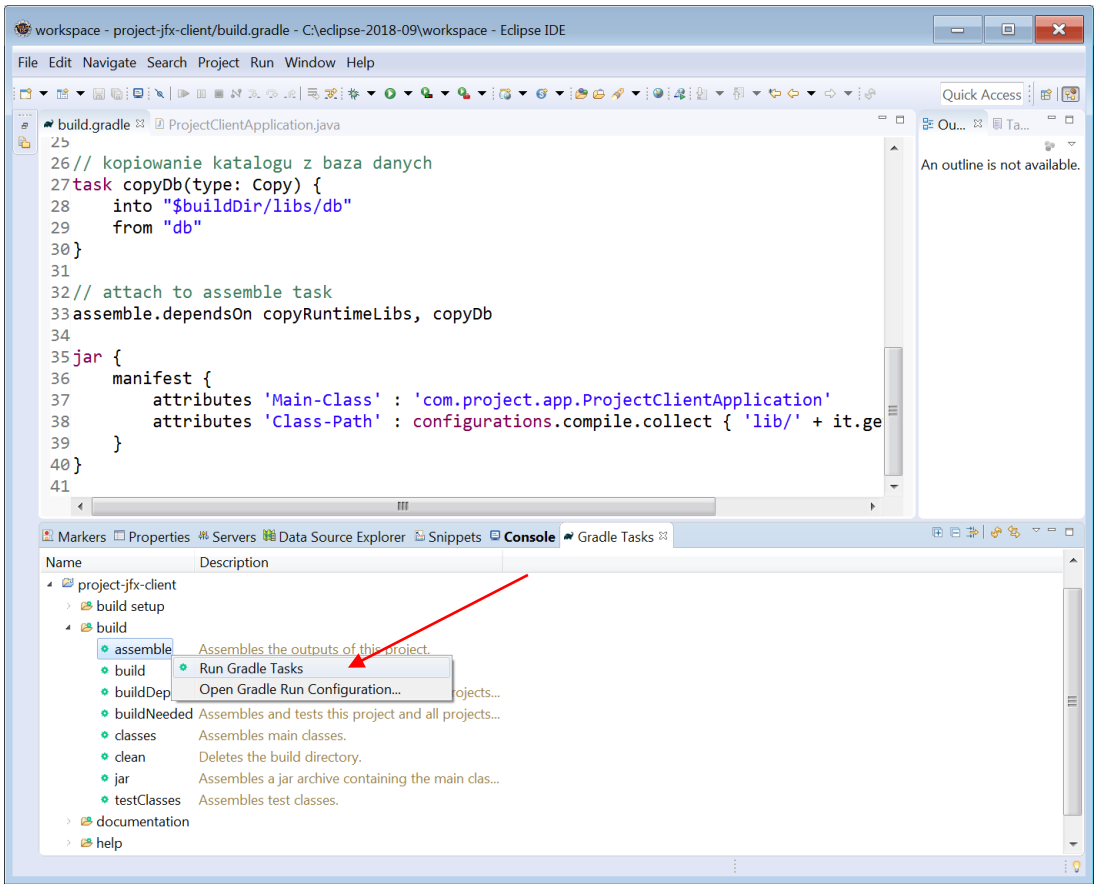
Zadania dodatkowe, tylko dla chętnych: implementacja metod *onActionBtnPierwsza* i *onActionBtnOstatnia*, zmiana rozmiaru strony, a także zaawansowane wyszukiwanie uwzględniające nazwę, identyfikator projektu oraz jego datę oddania. Do wyznaczania sposobu wyszukiwania można skorzystać z wyrażeń regularnych testujących zawartość pola tekstowego *txtSzukaj*.

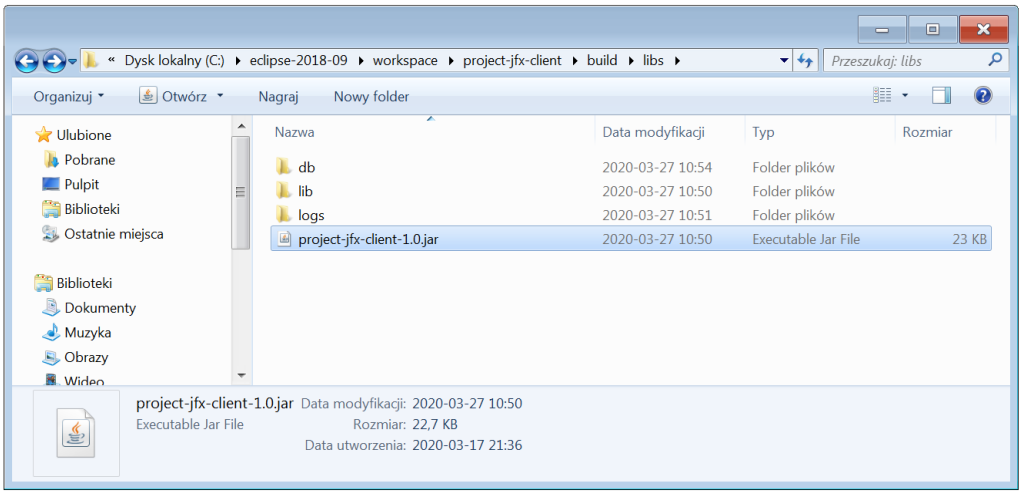
6. UTWORZENIE WERSJI URUCHOMIENIOWEJ

6.1. Do pliku *build.gradle* dodaj poniższą treść. Kliknij prawym przyciskiem myszki na głównej ikonce projektu i wybierz *Gradle -> Refresh Gradle Project*.

```
plugins {
 id 'java'
}

group = 'com.project'
version = '1.0'

repositories {
    jcenter()
}

dependencies {
       compile group: 'org.hsqldb', name: 'hsqldb', version: '2.5.0'
       compile group: 'org.hibernate', name: 'hibernate-core', version: '5.4.13.Final'
       compile group: 'ch.qos.logback', name: 'logback-classic', version: '1.2.3'
       compile group: 'ch.qos.logback', name: 'logback-core', version: '1.2.3'
       compile group: 'org.slf4j', name: 'slf4j-api', version: '1.7.30'
}

// kopiowanie katalogu bibliotekami
task copyRuntimeLibs(type: Copy) {
    into "$buildDir/libs/lib/"
    from configurations.compile
}

// kopiowanie katalogu z baza danych
task copyDb(type: Copy) {
    into "$buildDir/libs/db"
    from "db"
}

// attach to assemble task
assemble.dependsOn copyRuntimeLibs, copyDb

jar {
    manifest {
        attributes 'Main-Class' : 'com.project.app.ProjectClientApplication'
        attributes 'Class-Path' : configurations.compile.collect { 'lib/' + it.getName() }.join(' ')
    }
}
```

6.2. W widoku *Gradle Tasks* (jeżeli widok nie jest wyświetlany w Eclipse to z menu wybierz *Window -> Show View -> Other*, a następnie zaznacz *Gradle Tasks*) kliknij prawym przyciskiem myszki na *assemble* (z *project-jfx-client/build*) i wybierz *Run Gradle Tasks*. Utworzoną wersję uruchomieniową można znaleźć w katalogu *project-jfx-client\build\libs* (niewidocznym z poziomu Eclipse).


workspace - project-jfx-client/build.gradle - C:\eclipse-2018-09\workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Quick Access
build.gradle
ProjectClientApplication.java
// kopiowanie katalogu z baza danych
task copyDb(type: Copy) {
into "$buildDir/libs/db"
from "db"
}
// attach to assemble task
assemble.dependsOn copyRuntimeLibs, copyDb
jar {
manifest {
attributes 'Main-Class' : 'com.project.app.ProjectClientApplication'
attributes 'Class-Path' : configurations.compile.collect { 'lib/' + it.ge
}
}
An outline is not available.
Markers Properties Servers Data Source Explorer Snippets Console Gradle Tasks
Name
Description
project-jfx-client
build setup
build
assemble
Assembles the outputs of this project.
build
Run Gradle Tasks
Open Gradle Run Configuration...
buildNeeded
Assembles and tests this project and all projects...
classes
Assembles main classes.
clean
Deletes the build directory.
jar
Assembles a jar archive containing the main clas...
testClasses
Assembles test classes.
documentation
help


Dysk lokalny (C:) ▸ eclipse-2018-09 ▸ workspace ▸ project-jfx-client ▸ build ▸ libs ▸
Przeszukaj: libs
Organizuj
Otwórz
Nagraj
Nowy folder
Ulubione
Pobrane
Pulpit
Biblioteki
Ostatnie miejsca
Biblioteki
Dokumenty
Muzyka
Obrazy
Wideo
Nazwa
Data modyfikacji
Typ
Rozmiar
db
2020-03-27 10:54
Folder plików
lib
2020-03-27 10:50
Folder plików
logs
2020-03-27 10:51
Folder plików
project-jfx-client-1.0.jar
2020-03-27 10:50
Executable Jar File
23 KB
project-jfx-client-1.0.jar Data modyfikacji: 2020-03-27 10:50
Executable Jar File
Rozmiar: 22,7 KB
Data utworzenia: 2020-03-17 21:36

7. TESTOWANIE RELACJI

7.1. Edytuj plik *build.gradle* i w sekcji *dependencies* dodaj zależności JUnit.

```
dependencies {
    ...

    testImplementation group: 'org.junit.jupiter', name: 'junit-jupiter-api', version: '5.6.1'
    testRuntimeOnly group: 'org.junit.jupiter', name: 'junit-jupiter-engine', version: '5.6.1'
}
```

Poza tym na końcu tego pliku dodaj fragment włączający natywne wsparcie Gradle'a dla JUnit.

```
test {
    useJUnitPlatform() //aktywacja natywnego wsparcia (od wersji 4.6 Gradle'a)
    testLogging {
        showStandardStreams = true //włącza drukowanie komunikatów w konsoli
     }
 }
```

Następnie edytuj plik *project-jfx-client\gradle\wrapper\gradle-wrapper.properties* i jeżeli w parametrze *distributionUrl* jest podana starsza wersja od 4.6. to zaktualizuj wpis.

```
distributionUrl=https\://services.gradle.org/distributions/gradle-4.6-bin.zip
```

Kliknij prawym przyciskiem myszki na głównej ikonce projektu i wybierz *Gradle -> Refresh Gradle Project*.

7.2. W katalogu *src**test**\resources\META-INF* (utwórz nieistniejące podkatalogi) dodaj kolejny deskryptor utrwalania *persistence.xml*, który będzie wykorzystywany wyłącznie do testów. W tym przypadku będziemy używać bazę HSQLDB w trybie *In-Memory*.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
version="2.1">
   <persistence-unit name="testHsqlManager" transaction-type="RESOURCE_LOCAL">
      <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
      <class>com.project.model.Projekt</class>
      <class>com.project.model.Zadanie</class>
      <class>com.project.model.Student</class>
      <properties>
         <property name="javax.persistence.jdbc.driver" value="org.hsqldb.jdbcDriver" />
         <property name="javax.persistence.jdbc.user" value="admin" />
         <property name="javax.persistence.jdbc.password" value="admin" />
         <property name="javax.persistence.jdbc.url" value="jdbc:hsqldb:mem:projekty" />
         <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect" />
         <property name="hibernate.hbm2ddl.auto" value="create" />
         <property name="hibernate.show_sql" value="true" />
      </properties>
   </persistence-unit>
</persistence>
```

7.3. W *src\test\java* utwórz pakiet *com.project.test* i dodaj do niego klasę *ProjektTest*. Zaimplementuj puste metody testowe przedstawione poniżej. Po uruchomieniu testów sprawdź wydruki w konsoli.

Użyta konfiguracja pozwala na drukowanie komunikatów w konsoli i określa kolejność wykonywania metod testujących. Chociaż nie jest to standardowa praktyka stosowana w testowaniu, ani też typowy test jednostkowy, to z pewnością jego implementacja i analiza będą pomocne podczas wdrażania w tematykę mapowania obiektowo-relacyjnego.

```java
package com.project.test;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import java.io.FileNotFoundException;
import java.sql.SQLException;
import java.time.LocalDate;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInfo;
import org.junit.jupiter.api.TestMethodOrder;
import com.project.model.Projekt;
import com.project.model.Zadanie;
import com.project.model.Student;

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class ProjektTest {
    private static EntityManagerFactory entityManagerFactory;
    private static EntityManager entityManager;

    @BeforeAll
    public static void init() throws FileNotFoundException, SQLException {
        entityManagerFactory = Persistence.createEntityManagerFactory("testHsqlManager");
        entityManager = entityManagerFactory.createEntityManager();
    }

    @AfterAll
    public static void afterAll() {
        entityManager.close();
        entityManagerFactory.close();
    }

    // --- URUCHAMIANIE TESTÓW ---
    // ABY URUCHOMIĆ TESTY KLIKNIJ NA NAZWIE KLASY PRAWYM PRZYCISKIEM
    // MYSZY I WYBIERZ Z MENU 'Run As' -> 'Gradle Test' LUB PO USTAWIENIU
    // KURSORA NA NAZWIE KLASY WCIŚNIJ SKRÓT 'CTRL+ALT+X' A PÓŹNIEJ 'G'
    // MOŻNA RÓWNIEŻ ANALOGICZNIE URUCHAMIAĆ POJEDYNCZE METODY KLIKAJĄC
    // WCZEŚNIEJ NA ICH NAZWĘ
    // PO ZAKOŃCZENIU TESTÓW W PODKATALOGU PROJEKTU DOSTĘPNY JEST SZCZEGÓŁOWY RAPORT
    //    project-jpa-client\build\reports\tests\test\index.html,
    // JEGO ADRES DRUKOWANY JEST W KONSOLI, GDY JAKIŚ TEST ZAKOŃCZY SIĘ BŁĘDEM NP.
    //    file:///C:/eclipse-2018-09/workspace/project-jfx-client/build/reports/tests/test/index.html


   @Test
   @Order(1)
   public void dodawanieProjektuZZadaniami() {
      Projekt projekt = new Projekt("Aplikacji webowa", "Aplikacja w Javie", LocalDate.of(2020, 6, 19));

      Zadanie zadanie1 = new Zadanie("Instalacja kontenera serwletów", "Instalacja serwera Tomcat 9.0.33", 1);
      Zadanie zadanie2 = new Zadanie("Implementacja aplikacji", "Zgodna z wzorcem MVC", 2);

      //przypisujemy do zadań projekt
      zadanie1.setProjekt(projekt);
      zadanie2.setProjekt(projekt);

      entityManager.getTransaction().begin();
      entityManager.persist(zadanie1);      //utrwalanie zawsze dla wszystkich obiektów - projektu i jego zadań
      entityManager.persist(zadanie2);
      entityManager.persist(projekt);
      entityManager.getTransaction().commit();

      entityManager.refresh(projekt); //odświeżenie stanu zarządzanej encji
                                      //na podstawie informacji z bazy danych

      //sprawdzamy czy w bazie danych do projektu zostały przypisane zadania
      List<Zadanie> zadania = projekt.getZadania();
```

```
    assertNotNull(zadania);
    assertEquals(2, zadania.size());

    System.out.printf("Projekt - Id: %d, Nazwa: %s%n", projekt.getProjektId(), projekt.getNazwa());
    for (Zadanie zad : zadania) {
        System.out.printf(" Zadanie - Id: %d, Nazwa: %s%n", zad.getZadanieId(), zad.getNazwa());
    }
}
@Test
@Order(2)
public void usuwanieProjektuZZadaniami() {
    // TODO
    // Pamiętaj, że dane w powiązanych tabelach naszego modelu nie będą
    // automatycznie modyfikowane (ON DELETE NO ACTION, ON UPDATE NO ACTION).
    // Przy każdej próbie usuwania, czy modyfikacji projektu, do którego są
    // odwołania przez wartości kluczy obcych, zawsze generowany będzie błąd,
    // a polecenie DELETE lub UPDATE wycofywane.
    // Należy zatem zacząć od usuwania zadań.
}

@Test
@Order(3)
public void dodawanieProjektuZeStudentamiIZadaniami() {
    // TODO

}

@Test
@Order(4)
public void wyszukiwanieProjektuZeStudentamiIZadaniemTomcat() {
    // TODO
    // Pobierz projekty, do których zostało przypisanych co najmniej dwóch studentów
    // i które maja w nazwie lub opisie jakiegokolwiek zadania słowo 'Tomcat'.
}

@BeforeEach
public void before(TestInfo testInfo) {
    System.out.printf("-- METODA -> %s%n", testInfo.getTestMethod().get().getName());
}

@AfterEach
public void after(TestInfo testInfo) {
    System.out.printf("<- KONIEC -- %s%n", testInfo.getTestMethod().get().getName());
}
```