Platformy Technologiczne
Laboratorium 5
Java: JPA

Należy zaimplementować aplikację pozwalającą na zarządzanie katalogiem przechowywanym w relacyjnej bazie danych. Struktura katalogu zostanie podana przez prowadzącego na zajęciach.

Przykład:

```
public class Book {
	private Integer id;
	private String title;
	private List<Author> authors;
}

public class Author {
	private Integer id;
	private String name;
	private String surname;
	private Book book;
}
```

Encjami wykorzystywanymi w katalogu będą dwie klasy. Element przechowywany bezpośrednio w katalogu (np. ***Book***) oraz jego pod elementy (np. ***Author***). Elementy powinny być w relacji jeden-wiele.

Wczytywanie/zapisywanie z/do relacyjnej bazy realizuje się poprzez obiekt ***EntityManager*** stworzony za pomocą fabryki ***EntityManagerFactory***, która jest tworzona na podstawie deskryptora ***persistence.xml***. Deskryptor zawiera nazwę persistence unit (PU), konfiguracją sterownika do bazy danych, dostawcy implementacji JPA, konfigurację połączenia z bazą danych oraz listę klas będących encjami. Przy tworzeniu fabryki ***EntityManagerFactory*** należy pamiętać o użyciu tej samej nazwy PU jak ta użyta w deskryptorze.

Przykład:

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("BooksPU");
EntityManager em = emf.createEntityManager();
```

Wszystkie modyfikacje w bazie danych powinny być realizowane w ramach poprawnie obsłużonej transakcji. Zmiany takie jak utrwalenie nowego obiektu, aktualizacja obiektu, usunięcie obiektu realuzuje się przez metody obiektu ***EntityManger***: ***persist***, ***merge***, ***remove***. Należy zwrócić uwagę że argumentem metody ***remove*** powinien być obiekt zarządzany, czyli np. taki zwrócony przez metodę ***merge***.

Przykład:

```
try {
	em.getTransaction().begin();
	em.remove(em.merge(book));
	em.getTransaction().commit();
} catch (Exception ex) {
	if (em.getTransaction().isActive()) {
		em.getTransaction().rollback();
	}
}
```

Aby wczytać wszystkie elementy z bazy danych można skorzystać z zapytania zapisanego w JPQL. Jest to niezależny od implementacji sterownika bazy danych, język zapytań bazujący na encjach. Należy zwrócić uwagę, że budując zapytanie zamiast nazwami tabel i kolumn należy posługiwać się nazwami klas i własności.

Przykład:

```
List<Book> books = em.createQuery("select b from Book b").getResultList();
```

Aplikacja musi korzystać z komponentu TableView z JavaFX. Zarówno kolumny jak i sposób ich wyświetlania można zdefiniować w pliku fxml korzystając z odpowiednich obiektów ***TableColumn*** i ***PropertyValueFactory*** oraz odpowiednich własności ***columns*** i ***cellValueFactory***.

Przykład:

```
<TableView fx:id="tableView">
	<columns>
		<TableColumn fx:id="titleColumn">
			<cellValueFactory>
				<PropertyValueFactory property="title"/>
			</cellValueFactory>
		</TableColumn>
	</columns>
</TableView>
```

Wszystkie atrybuty (np. ***title***) muszą być edytowalne z poziomu tabelki. W tym celu obiekt tabeli musi być ustawiony jako edytowalny a do każdej z kolumn należy podpiąć odpowiednie obiekty ***TableCell*** poprzez własność ***cellFactory***. Jako obiektu ***TableCell*** należy skorzystać ze standardowego ***TextFieldTableCell*** renderującego komórkę tabeli jako zwykłe pole tekstowe. Należy zwrócić uwagę że ustawienie obiektu ***TextFieldTableCell*** pozwala jedynie na edycję zawartości komórki. Należy dopisać odpowiedni handler (klasa implementująca interfejs ***EventHandler***) obsługujący przepisanie nowej wartości do odpowiedniego obiektu.

Przykład:

```
TableView.setEditable(true);

titleColumn.setCellFactory(TextFieldTableCell.forTableColumn());
titleColumn.setOnEditCommit(new EventHandler<CellEditEvent<Book, String>>() {
	@Override
	public void handle(CellEditEvent<Mage, String> t) {
		Book book = t.getRowValue();
		String newTitle = t.getNewValue();
		book.setTitle(newTitle);
	}
});
```

W przypadku pół typu innego niż ***String*** (np. ***int***, ***double***, itd.) należy skorzystać z odpowiedniego konwertera (np. ***IntegerStringConverter***, ***DoubleStringConverter***, itd.).

Przykład:

```
IntegerStringConverter conv = new IntegerStringConverter();
pagesColumn.setCellFactory(TextFieldTableCell.forTableColumn(conv));
```

Aplikacja powinna wyświetlać dwie tabele. Pierwsza wyświetla elementy pobrane bezpośrednio z bazy danych (np. ***Book***), a druga pod elementy (np. ***Author***) wybranego elementu. Można to zrealizować

przez poprawne wykrycie zdarzenia zmiany zaznaczenia w pierwszej tabeli i ustawienie elementów dla drugiej.

Przykład:

```
@FXML
private TableView<Book> booksTableView;

@FXML
private TableView<Author> authorsTableView;

ListProperty<Author> authors = new SimpleListProperty<>();

private class BookChangeListener implements ChangeListener<Book>() {
	@Override
	public void changed(ObservableValue<? extends Book> value, Book oldV, Book newV) {
		if (newV != null) {
			authors.set((ObservableList<Author>) newV.getAuthors());
		} else {
			authors.setValue(null);
		}
	}
}
```

```
authorsTableView.itemsProperty().bind(authors);

booksTableView.getSelectionModel().selectedItemProperty().
		addListener(new BookChangeListener());
```

Aplikacja musi pozwalać (np. za pomocą przycisków) na dodawanie nowego, usuwanie i edycję elementu. Wszystkie akcje powinny wykonywać odpowiednie operacje na obiekcie ***EntityManager***.

Punktacja:

- wczytanie katalogu z bazy danych i wyświetlenie go w tabeli: 1 pkt,
- wyświetlenie katalogu w dwóch tabelach (elementy i pod elementy): 1 pkt,
- usuwanie elementów z tabeli i bazy danych: 1 pkt,
- dodawanie nowych elementów do tabeli i bazy danych: 1 pkt,
- edycja elementów w tabeli i bazie danych: 1 pkt.

W przypadku realizacji usuwania, dodawania i edycji tylko na elementach pobranych bezpośrednio z katalogu (np. poprzez realizację tylko jednej tabeli wyświetlającej obiekty klasy ***Book***) przysługuje połowa punktów.