

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФРАСТРУКТУРИ ТА ТЕХНОЛОГІЙ**  
**ІНСТИТУТ УПРАВЛІННЯ, ТЕХНОЛОГІЙ ТА ПРАВА**  
**ФАКУЛЬТЕТ УПРАВЛІННЯ І ТЕХНОЛОГІЙ**

Кафедра інформаційних технологій

**Лабораторна робота №2**

з дисципліни «Доменна інженерія»

з теми: «Apache Jena: створення застосунку та виконання SPARQL-запитів до  
онтології»

Варіант 13

**Виконав:**

Студент групи

ІПЗд-23121 маг.

Петренко Д.М.

**Перевірив:**

Доцент кафедри ІТ

Ткаченко О.А.

## Практична робота №2

**Тема:** Apache Jena: створення застосунку та виконання SPARQL-запитів до онтології.

### Завдання:

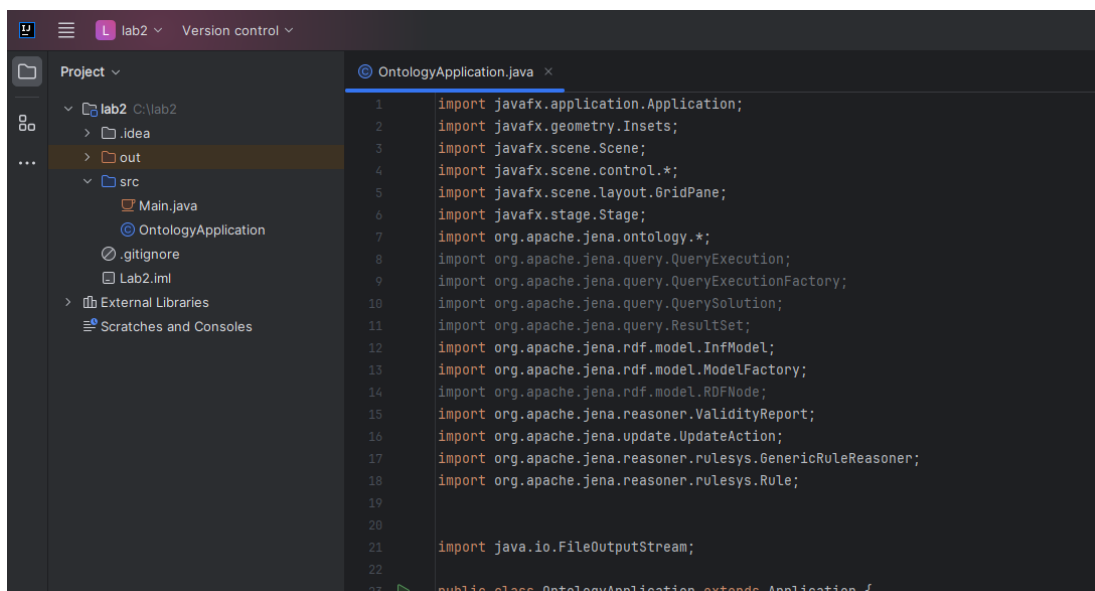
З використанням мови Java та Apache Jena API розробити Java-застосунок з інтерфейсом користувача для роботи з онтологічними моделями. Вставити у звіт код java-застосунку та скріншоти результатів його роботи.

Створіть java-застосунок і імпортуйте в нього свою онтологію, розроблену у Практичній роботі 1. Програмно виконайте такі дії з вашою онтологією:

1. перевірте свою онтологію за допомогою Reasoner;
2. додайте ще один клас до онтології;
3. додайте індивідуал до обраного класу;
4. створіть інтерфейс користувача (не знайомого зі SPARQL) для виконання запитів INSERT DATA до вашої онтології.

### ХІД РОБОТИ:

Створимо Джава застосунок, котрий буде виконувати функції вище. Зайдемо в IntelliJ IDEA створимо дану програму.



```
Project
├── lab2
│   ├── .idea
│   ├── out
│   └── src
│       ├── Main.java
│       └── OntologyApplication
├── .gitignore
├── Lab2.iml
├── External Libraries
└── Scratches and Consoles

OntologyApplication.java
1  import javafx.application.Application;
2  import javafx.geometry.Insets;
3  import javafx.scene.Scene;
4  import javafx.scene.control.*;
5  import javafx.scene.layout.GridPane;
6  import javafx.stage.Stage;
7  import org.apache.jena.ontology.*;
8  import org.apache.jena.query.QueryExecution;
9  import org.apache.jena.query.QueryExecutionFactory;
10 import org.apache.jena.query.QuerySolution;
11 import org.apache.jena.query.ResultSet;
12 import org.apache.jena.rdf.model.InfModel;
13 import org.apache.jena.rdf.model.ModelFactory;
14 import org.apache.jena.rdf.model.RDFNode;
15 import org.apache.jena.reasoner.ValidityReport;
16 import org.apache.jena.update.UpdateAction;
17 import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
18 import org.apache.jena.reasoner.rulesys.Rule;
19
20
21 import java.io.FileOutputStream;
22
23 public class OntologyApplication extends Application {
```

## Опишемо код програмного застосунку.

Підключаємо бібліотеки JavaFX та Apache Jena:

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import org.apache.jena.ontology.*;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.reasoner.ValidityReport;
import org.apache.jena.update.UpdateAction;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
import org.apache.jena.reasoner.rulesys.Rule;

import java.io.FileOutputStream;
```

### JavaFX Imports:

- **import javafx.application.Application;**: Включає клас **Application**, який є базовим класом для всіх JavaFX додатків.
- **import javafx.geometry.Insets;**: Включає клас **Insets**, який представляє відступи в JavaFX для вирівнювання елементів у контейнерах.
- **import javafx.scene.Scene;**: Включає клас **Scene**, який представляє сцену в JavaFX, що містить всі елементи інтерфейсу користувача.
- **import javafx.scene.control.\*;**: Включає клас **Control** та його підкласи, які представляють різні елементи управління (кнопки, тексти, тощо).
- **import javafx.scene.layout.GridPane;**: Включає клас **GridPane**, який використовується для створення сітки елементів у JavaFX інтерфейсі.

### Apache Jena Imports:

- **import org.apache.jena.ontology.\*;**: Включає класи для роботи з онтологіями, такі як **OntModel**, **OntModelSpec** і інші.
- **import org.apache.jena.query.\*;**: Включає класи для виконання SPARQL-запитів, такі як **QueryExecution**, **QueryExecutionFactory**, **QuerySolution**, **ResultSet**.

- **import org.apache.jena.rdf.model.\*;**: Включає класи для роботи з RDF-моделлю, такі як **RDFNode** і **ModelFactory**.
- **import org.apache.jena.reasoner.\*;**: Включає класи для роботи з резонерами, такі як **ValidityReport**.
- **import org.apache.jena.update.\*;**: Включає класи для виконання SPARQL-запитів UPDATE, такі як **UpdateAction**.
- **import org.apache.jena.reasoner.rulesys.\*;**: Включає класи для роботи з правилами резонера, такі як **GenericRuleReasoner** і **Rule**.
- **import java.io.FileOutputStream;**: Включає клас **FileOutputStream** для запису RDF-моделі у файл.

Оголошення класу та об'єктів, які будуть використовуватись в програмі:

```
public class OntologyApplication extends Application {

    private TextArea outputTextArea; // Оголошення об'єкту для виводу тексту
    private TextField addClassTextField; // Оголошення поля для введення нового класу
    private TextField addIndividualClassTextField; // Оголошення поля для введення
класу, до якого буде доданий індивід
    private TextField addIndividualTextField; // Оголошення поля для введення нового
індивіда
    private TextField individualTextField; // Оголошення поля для введення імені
індивіда
    private TextField objectiveTextField; // Оголошення поля для введення hasObjective
    private TextField methodologyTextField; // Оголошення поля для введення
hasMethodology
    private TextField dateTextField; // Оголошення поля для введення hasYear

    private OntModel model; // Оголошення онтологічної моделі

    public static void main(String[] args) { Launch(args); } // Запуск додатку
}
```

Розробка інтерфейсу, кнопок та текстових полей:

```
public void start(Stage primaryStage) {
    primaryStage.setTitle("ЛАБ2_ДІ_Онтологія"); // Встановлює заголовок головного вікна
додатку

    // Ініціалізація OntModel (онтологічна модель)
    model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

    // Створення компонентів користувацького інтерфейсу
    outputTextArea = new TextArea(); // Текстова область для виводу інформації,
недоступна для редагування
    outputTextArea.setEditable(false);

    // Завдання 1
    addClassTextField = new TextField(); // Поле введення для додавання нового класу
    Button addClassButton = new Button("Add Class"); // Кнопка для виклику методу
додавання класу при натисканні
    addClassButton.setOnAction(e -> addClass());
}
```

```

// Завдання 2
addIndividualClassTextField = new TextField();// Поле введення для класу, до якого
буде доданий індивід
addIndividualTextField = new TextField();// Поле введення для додавання нового
індивіда
Button addIndividualButton = new Button("Add Individual");// Кнопка для виклику
методу додавання індивіда при натисканні
addIndividualButton.setOnAction(e ->
addIndividual(addIndividualClassTextField.getText(),
addIndividualTextField.getText()));

// Завдання 3
individualTextField = new TextField();// Поле введення для імені індивіда
objectiveTextField = new TextField();// Поле введення для мети
methodologyTextField = new TextField();// Поле введення для методології
dateTextField = new TextField();// Поле введення для дати
Button addDataButton = new Button("INSERT DATA");// Кнопка для виклику методу
вставки даних при натисканні
addDataButton.setOnAction(e ->
executeInsertDataQuery(individualTextField.getText(), objectiveTextField.getText(),
methodologyTextField.getText(), dateTextField.getText()));

// Налаштування розмітки (GridPane) для розташування компонентів
GridPane gridPane = new GridPane();
gridPane.setHgap(10);
gridPane.setVgap(10);
gridPane.setPadding(new Insets(10, 10, 10, 10));

gridPane.add(outputTextArea, 0, 0, 2, 1);

gridPane.add(new Label("Add Class:"), 0, 1);
gridPane.add(addClassTextField, 1, 1);
gridPane.add(addClassButton, 2, 1);

gridPane.add(new Label("Add Individual to Class:"), 0, 2);
gridPane.add(addIndividualClassTextField, 1, 2);
gridPane.add(addIndividualTextField, 2, 2);
gridPane.add(addIndividualButton, 3, 2);

gridPane.add(new Label("Individual:"), 0, 4);
gridPane.add(individualTextField, 1, 4);
gridPane.add(new Label("Objective:"), 0, 5);
gridPane.add(objectiveTextField, 1, 5);
gridPane.add(new Label("Methodology:"), 0, 6);
gridPane.add(methodologyTextField, 1, 6);
gridPane.add(new Label("Date:"), 0, 7);
gridPane.add(dateTextField, 1, 7);
gridPane.add(addDataButton, 3, 3);

// Налаштування сцени з використанням створеної розмітки
Scene scene = new Scene(gridPane, 600, 400); // Розміри сцени 600x400 пікселів

// Завантаження онтології
loadOntology();

// Налаштування головного вікна
primaryStage.setScene(scene);
primaryStage.show();
}

```

Далі завантажуюмо онтологію:

```
private void loadOntology() {
    // Шлях до файлу онтології
    String ontologyFilePath = "C:/Users/Den4ik/Desktop/LAB1.rdf";

    try {
        // Завантаження онтології з файлу
        model.read(ontologyFilePath);

        // Виведення повідомлення про успішне завантаження
        outputTextArea.appendText("Ontology Loaded Successfully.\n");

        // Налаштування GenericRuleReasoner
        String rules = "[rule1: (?a rdf:type ?b) (?b rdf:type ?c) -> (?a rdf:type ?c)]"; // Додавання інших правил за необхідності
        GenericRuleReasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rules));

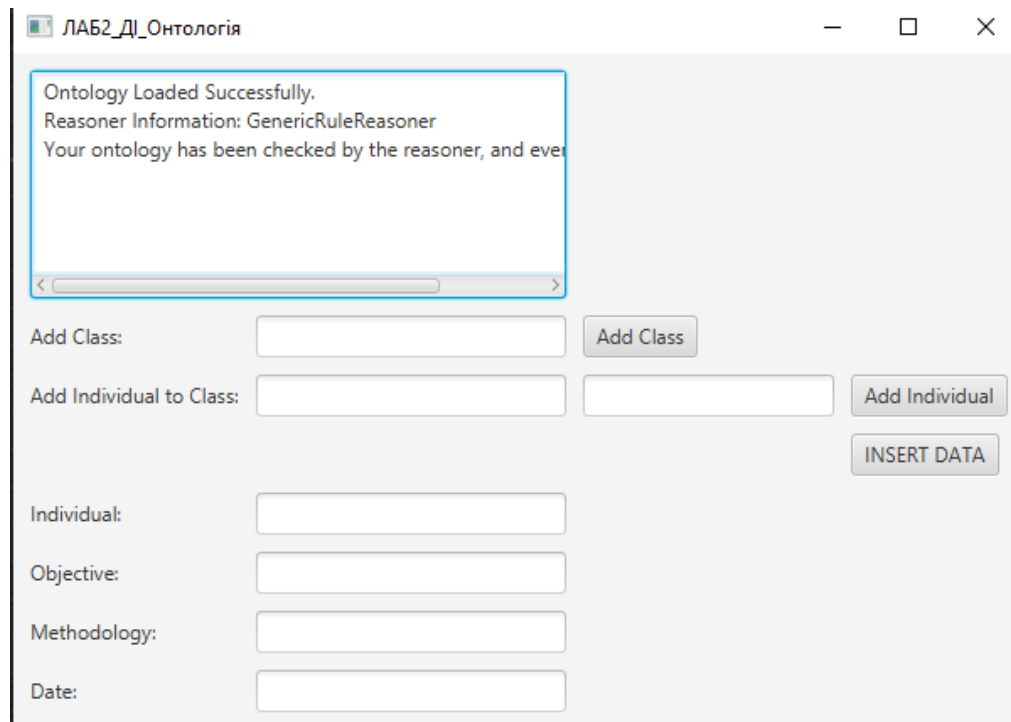
        InfModel infModel = ModelFactory.createInfModel(reasoner, model);

        // Виведення інформації про резонер
        outputTextArea.appendText("Reasoner Information: GenericRuleReasoner\n");

        // Виклик validate() для перевірки онтології
        ValidityReport validityReport = infModel.validate();

        if (validityReport == null || validityReport.isValid()) {
            // Виведення повідомлення про успішну перевірку
            outputTextArea.appendText("Your ontology has been checked by the reasoner, and everything is okay.\n");
        } else {
            // Виведення повідомлення про невірну онтологію та відображення помилок валідації
            outputTextArea.appendText("Ontology is not valid.\n");
            outputTextArea.appendText("Validation Errors:\n" + validityReport);
        }
    } catch (Exception e) {
        // Виведення повідомлення про помилку при завантаженні
        outputTextArea.appendText("Failed to load ontology.\n");
        e.printStackTrace();
    }
}
```

Результатом даного коду вище є такий застосунок написаний на мові програмування Java:



## Розглянемо функціонал кожної кнопки:

- Кнопка «Add Class» (Завдання 1):

```
private void addClass(String className) {
    // Оновлення простору імен онтології
    String ontologyNamespace =
"http://www.semanticweb.org/den4ik/ontologies/2023/11/lab1_di";

    // SPARQL-запит для додавання класу
    String sparqlQuery = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n"
+
    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
    "INSERT DATA {\n" +
    "  <" /*+ ontologyNamespace*/ + className + "> rdf:type rdfs:Class\n" +
    "}";

    try {
        // Створення об'єкта для виконання запиту
        UpdateAction.parseExecute(sparqlQuery, model);

        // Збереження моделі у файл
        model.write(new FileOutputStream("C:/Users/Den4ik/Desktop/LAB1.rdf"));

        // Виведення повідомлення про успішне додавання класу та оновлення онтології
        outputTextArea.appendText("Class '" + className + "' successfully added and
ontology updated.\n");
    } catch (Exception e) {
        // Виведення повідомлення про помилку
        outputTextArea.appendText("Failed to add class '" + className + "'.\n");
        e.printStackTrace();
    }
}

private void addClass() {
    String className = addClassTextField.getText(); // Отримання назви класу з
текстового поля
```

```
addClass(className); // Додавання класу до онтології за допомогою SPARQL
}
```

Цей код відповідає за додавання нового класу до онтології за допомогою мови запитів SPARQL. Розглянемо його по кроках:

`private void addClass(String className){...}` - цей метод призначений для додавання нового класу до онтології. Він отримує ім'я класу як вхідний параметр.

`UpdateAction.parseExecute(sparqlQuery, model);` - виконує SPARQL-запит, додаючи клас до моделі онтології.

`model.write(new FileOutputStream("C:/Users/Den4ik/Desktop/LAB1.rdf"));`  
- зберігає оновлену модель у файл.

`outputTextArea.appendText("Class '" + className + "' successfully added and ontology updated.\n");` - виводить повідомлення про успішне додавання класу та оновлення онтології.

`private void addClass()` - цей метод викликається при натисканні кнопки «Add Class» у користувацькому інтерфейсі. Він отримує ім'я класу з текстового поля і викликає метод `addClass(className)`, передаючи отримане ім'я.

- Кнопка «Add Individual» (Завдання 2):

```
private void addIndividual(String className, String individualName) {
    // Оновлення простору імен онтології
    String ontologyNamespace =
"http://www.semanticweb.org/den4ik/ontologies/2023/11/lab1_di";
    // SPARQL-запит для додавання індивіда до існуючого класу
    String sparqlQuery = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n"
+
    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
    "INSERT DATA {\n" +
    "  <" + individualName + "> rdf:type <" + className + ">\n" +
    "}";
    try {
        // Створення об'єкта для виконання запиту
        UpdateAction.parseExecute(sparqlQuery, model);
        // Збереження моделі у файл
        model.write(new FileOutputStream("C:/Users/Den4ik/Desktop/LAB1.rdf"));
        // Виведення повідомлення про успішне додавання індивіда до класу та оновлення
        онтології
        outputTextArea.appendText("Individual '" + individualName + "' added to class
'" + className + "' and ontology updated.\n");
    } catch (Exception e) {
        // Виведення повідомлення про помилку
        outputTextArea.appendText("Failed to add individual '" + individualName + "' to
class '" + className + "'.\n");
        e.printStackTrace();
    }
}
```



Цей код відповідає за додавання нового індивіда до існуючого класу в онтології за допомогою мови запитів SPARQL. Розглянемо його по кроках:

```
+private void addIndividual(String className, String individualName){...}
```

- цей метод призначений для додавання нового індивіда до існуючого класу в онтології. Він отримує ім'я класу та індивіда як вхідні параметри.

```
+model.write(new FileOutputStream("C:/Users/Den4ik/Desktop/LAB1.rdf"));
```

- зберігає оновлену модель у файл.

```
+UpdateAction.parseExecute(sparqlQuery, model);
```

 - виконує SPARQL-запит, додаючи індивіда до моделі онтології.

```
+outputTextArea.appendText("Individual '" + individualName + "' added to  
class '" + className + "' and ontology updated.\n");
```

 - виводить повідомлення про успішне додавання індивіда до класу та оновлення онтології.

Конструкція **try-catch** використовується для обробки можливих винятків під час виконання SPARQL-запиту. У разі виникнення помилки виводиться повідомлення про невдале додавання індивіда, і стек викликів помилки виводиться на консоль.

- *Кнопка «INSERT DATA» (Завдання 3):*

```
private void executeInsertDataQuery(String individual,String objective, String  
methodology, String date) {  
    // Перевірка, чи всі поля заповнені  
    if (individual.isEmpty() || objective.isEmpty() || methodology.isEmpty() ||  
date.isEmpty()) {  
        outputTextArea.appendText("Будь ласка, заповніть всі поля.\n");  
        return;  
    }  
  
    // Підготовка SPARQL INSERT DATA запиту  
    String insertDataQuery = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-  
ns#>\n" +  
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +  
        "PREFIX ex:  
<http://www.semanticweb.org/den4ik/ontologies/2023/11/lab1_di>\n" +  
        "INSERT DATA {\n" +  
        "    ex:" + individual + " rdf:type ex:MarketResearchStudy ;\n" +  
        "        ex:hasObjective '\"' + objective + "\"' ;\n" +  
        "        ex:hasMethodology '\"' + methodology + "\"' ;\n" +  
        "        ex:hasYear '\"' + date + "\"' .\n" +  
        "}";  
  
    try {  
        // Виконання SPARQL INSERT DATA запиту  
        UpdateAction.parseExecute(insertDataQuery, model);  
  
        // Збереження моделі у файл (за бажанням)  
        model.write(new FileOutputStream("C:/Users/Den4ik/Desktop/LAB1.rdf"));
```

```

        // Виведення повідомлення про успішне виконання
        outputTextArea.appendText("Дані успішно додані до онтології.\n");
    } catch (Exception e) {
        // Виведення повідомлення про помилку
        outputTextArea.appendText("Не вдалося виконати запит INSERT DATA.\n");
        e.printStackTrace();
    }
}

```

Цей код відповідає за виконання SPARQL INSERT DATA запиту для додавання нового індивіда (конкретно, об'єкта типу **ex:MarketResearchStudy**) до онтології з введеними користувачем даними. Розглянемо його крок за кроком:

✚ `private void executeInsertDataQuery(String individual, String objective, String methodology, String date)` - цей метод викликається для виконання SPARQL INSERT DATA запиту.

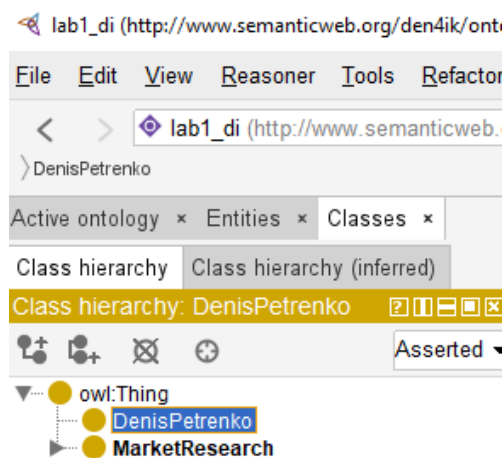
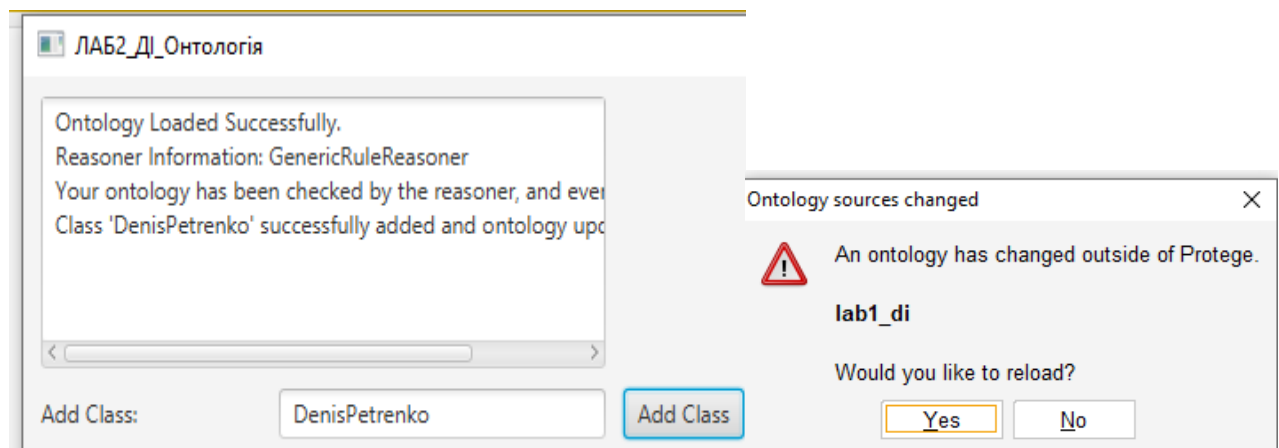
✚ Перш за все, проводиться перевірка, чи всі необхідні поля заповнені (індивід, мета, методологія, дата). Якщо хоча б одне поле порожнє, виводиться відповідне повідомлення, і виконання методу припиняється.

✚ Далі формується SPARQL INSERT DATA запит, де «ex:» - простір імен онтології, «individual» - назва індивіда, а «objective», «methodology», «date» - відповідно, значення для властивостей «hasObjective», «hasMethodology», «hasYear».

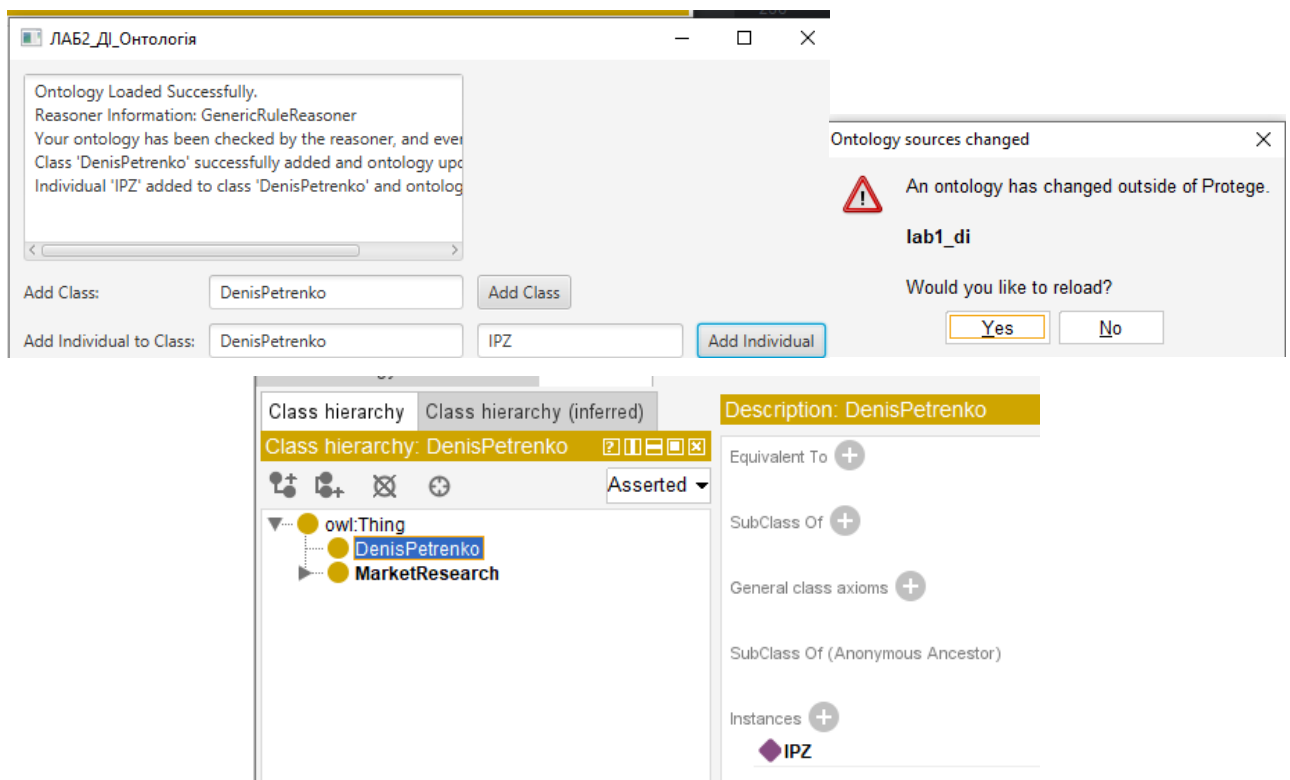
✚ `UpdateAction.parseExecute(insertDataQuery, model);` - виконує SPARQL INSERT DATA запит, додаючи новий індивід та його атрибути до моделі онтології.

## Покажемо принцип роботи програми:

### 1. Додаємо Клас:



### 2. Додаємо Індивідуал в Клас:



### 3. Виконання запитів INSERT DATA до вашої онтології:

ЛАБ2\_Ді\_Онтологія

Ontology Loaded Successfully.  
Reasoner Information: GenericRuleReasoner  
Your ontology has been checked by the reasoner, and e  
Class 'DenisPetrenko' successfully added and ontology c  
Individual 'IPZ' added to class 'DenisPetrenko' and ontol  
Дані успішно додані до онтології.

Add Class: DenisPetrenko Add Class

Add Individual to Class: DenisPetrenko IPZ Add Individual

Individual: MarketResearchStudy1

Objective: Evaluate market demand for ind

Methodology: Surveys and interviews with key

Date: 2023

INSERT DATA

Ontology sources changed

An ontology has changed outside of Protege.

lab1\_di

Would you like to reload?

Yes No

Active ontology x Entities x Classes x

Annotation properties Datatypes Individuals

Classes Object properties Data properties

Individuals: lab1\_diMarketResearchStudy1

lab1\_diMarketResearchStudy1

CompetitiveAnalysis1  
CompetitiveAnalysis2  
ConsumerStudy1  
ConsumerStudy2  
IndustrialStudy1  
IndustrialStudy2  
IPZ  
lab1\_diMarketResearchStudy1  
MarketResearchStudy1  
MarketResearchStudy2  
ProductStudy1  
ProductStudy2

lab1\_diMarketResearchStudy1 — http://www.semanticweb.org/den4ik/ontologies/2023/1

Annotations Usage

Annotations: lab1\_diMarketResearchStudy1

Annotations +

lab1\_dihhasMethodology  
Surveys and interviews with key stakeholders.

lab1\_dihhasObjective  
Evaluate market demand for industrial machinery.

lab1\_dihhasYear  
2023

Description: lab1\_diMarketRes Property assertions: lab1\_diMark

Types +

lab1\_diMarketResearch ? @ x o

Same Individual As +

Different Individuals +

Object property assertions +

Data property assertions +

Negative object property assertions +

http://www.semanticweb.org/den4ik/ontologies/2023/1/lab1\_di#ProductStudy2