|  |  |
| --- | --- |
|  | Министерство образования и науки Российской Федерации |
| Федеральное государственное автономное образовательное учреждение высшего образования |
| «Уральский федеральный университет имени первого Президента России Б. Н. Ельцина» |
| Институт радиоэлектроники и информационных технологий – РтФ |
|  |
|  | |

**ОТЧЕТ**

О ВЫПОЛНЕНИИ

ЛАБОРАТОНОЙ РАБОТЫ № 8

Руководитель ст. пр. Н. А. Архипов

Студент гр. РИМ-130971 Е.В. Треглазов

Екатеринбург 2024

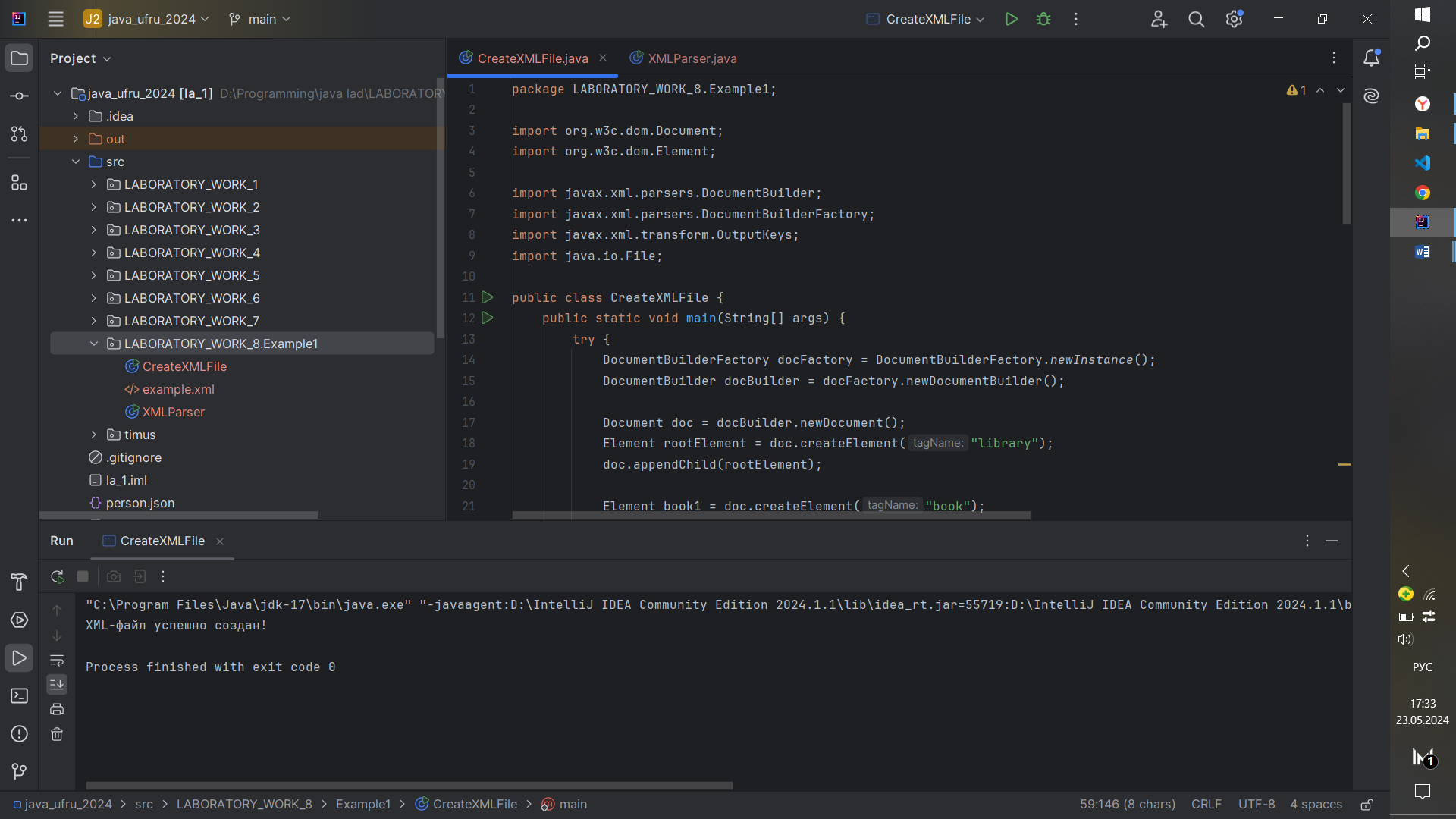
Репозиторий: https://github.com/kuk86kuk/java\_ufru\_2024.

Example1

1.

public class CreateXMLFile {  
 public static void main(String[] args) {  
 try {  
 DocumentBuilderFactory docFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder docBuilder = docFactory.newDocumentBuilder();  
  
 Document doc = docBuilder.newDocument();  
 Element rootElement = doc.createElement("library");  
 doc.appendChild(rootElement);  
  
 Element book1 = doc.createElement("book");  
 rootElement.appendChild(book1);  
  
 Element title1 = doc.createElement("title");  
 title1.appendChild(doc.createTextNode("Война и мир"));  
 book1.appendChild(title1);  
  
 Element author1 = doc.createElement("author");  
 author1.appendChild(doc.createTextNode("Лев Толстой"));  
 book1.appendChild(author1);  
  
 Element year1 = doc.createElement("year");  
 year1.appendChild(doc.createTextNode("1869"));  
 book1.appendChild(year1);  
  
 Element book2 = doc.createElement("book");  
 rootElement.appendChild(book2);  
  
 Element title2 = doc.createElement("title");  
 title2.appendChild(doc.createTextNode("Мастер и Маргарита"));  
 book2.appendChild(title2);  
  
 Element author2 = doc.createElement("author");  
 author2.appendChild(doc.createTextNode("Михаил Булгаков"));  
 book2.appendChild(author2);  
  
 Element year2 = doc.createElement("year");  
 year2.appendChild(doc.createTextNode("1967"));  
 book2.appendChild(year2);  
  
 doc.setXmlStandalone(true);  
 doc.normalizeDocument();  
 javax.xml.transform.TransformerFactory tf = javax.xml.transform.TransformerFactory.*newInstance*();  
 javax.xml.transform.Transformer transformer = tf.newTransformer();  
 transformer.setOutputProperty(OutputKeys.*ENCODING*, "UTF-8");  
 transformer.setOutputProperty(OutputKeys.*STANDALONE*, "yes");  
 transformer.setOutputProperty(OutputKeys.*INDENT*, "yes");  
 javax.xml.transform.dom.DOMSource source = new javax.xml.transform.dom.DOMSource(doc);  
 javax.xml.transform.stream.StreamResult result = new javax.xml.transform.stream.StreamResult(new File("src/LABORATORY\_WORK\_8/Example1/example.xml"));  
 transformer.transform(source, result);  
 System.*out*.println("XML-файл успешно создан!");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

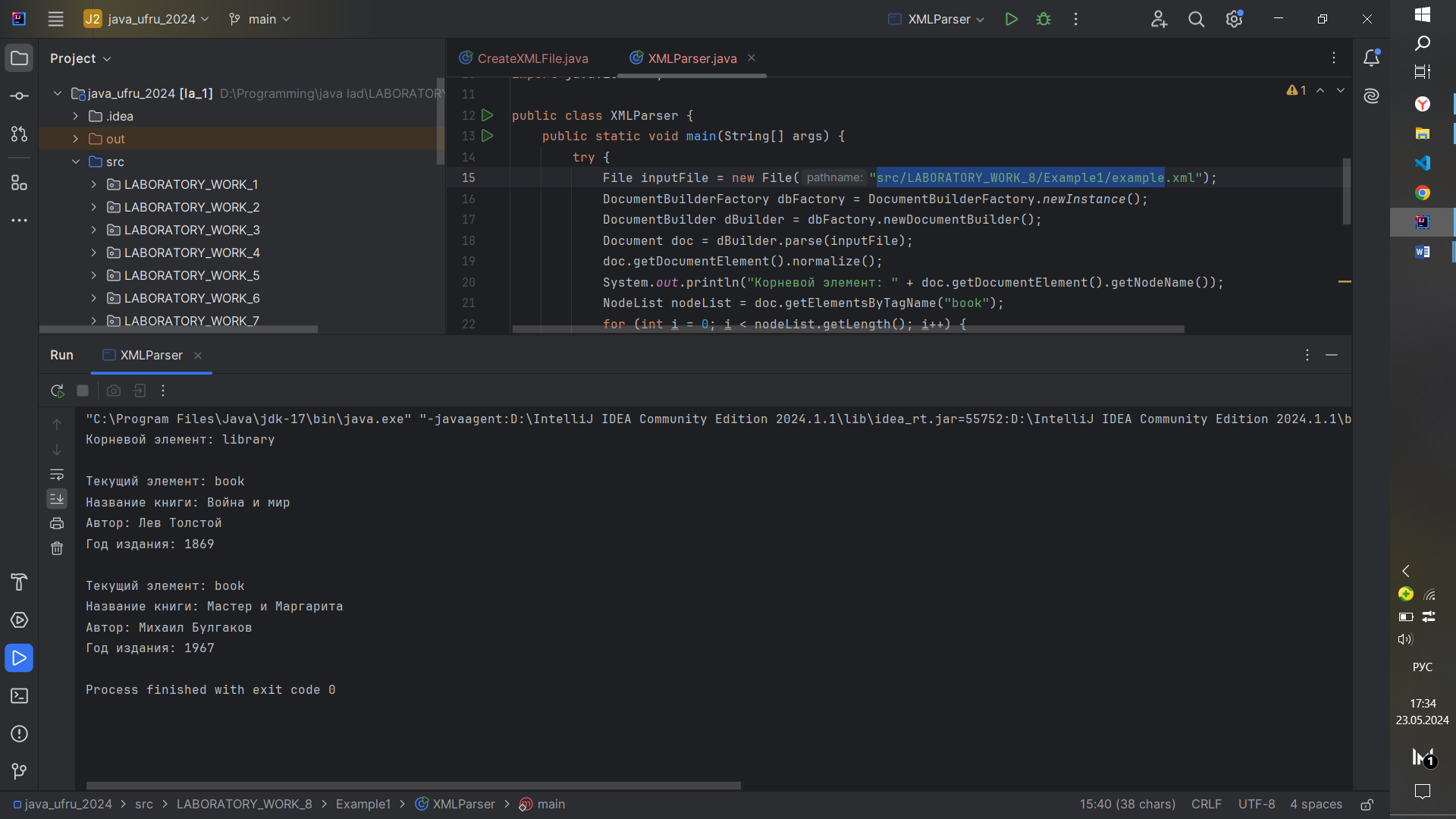
Мы начинаем с инициализации DocumentBuilderFactory и DocumentBuilder для получения Document, который будет корнем нашего XML. Далее, мы создаём элементы и встраиваем их в документ, используя методы createElement и appendChild. После построения XML, мы применяем Transformer для преобразования DOM-дерева в файл. Завершив запись, у нас есть XML-файл с данными, заданными в программе.



public class XMLParser {  
 public static void main(String[] args) {  
 try {  
 File inputFile = new File("src/LABORATORY\_WORK\_8/Example1/example.xml");  
 DocumentBuilderFactory dbFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();  
 Document doc = dBuilder.parse(inputFile);  
 doc.getDocumentElement().normalize();  
 System.*out*.println("Корневой элемент: " + doc.getDocumentElement().getNodeName());  
 NodeList nodeList = doc.getElementsByTagName("book");  
 for (int i = 0; i < nodeList.getLength(); i++) {  
 Node node = nodeList.item(i);  
 System.*out*.println("\nТекущий элемент: " + node.getNodeName());  
 if (node.getNodeType() == Node.*ELEMENT\_NODE*) {  
 Element element = (Element) node;  
 System.*out*.println("Название книги: " + element.getElementsByTagName("title").item(0).getTextContent());  
 System.*out*.println("Автор: " + element.getElementsByTagName("author").item(0).getTextContent());  
 System.*out*.println("Год издания: " + element.getElementsByTagName("year").item(0).getTextContent());  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

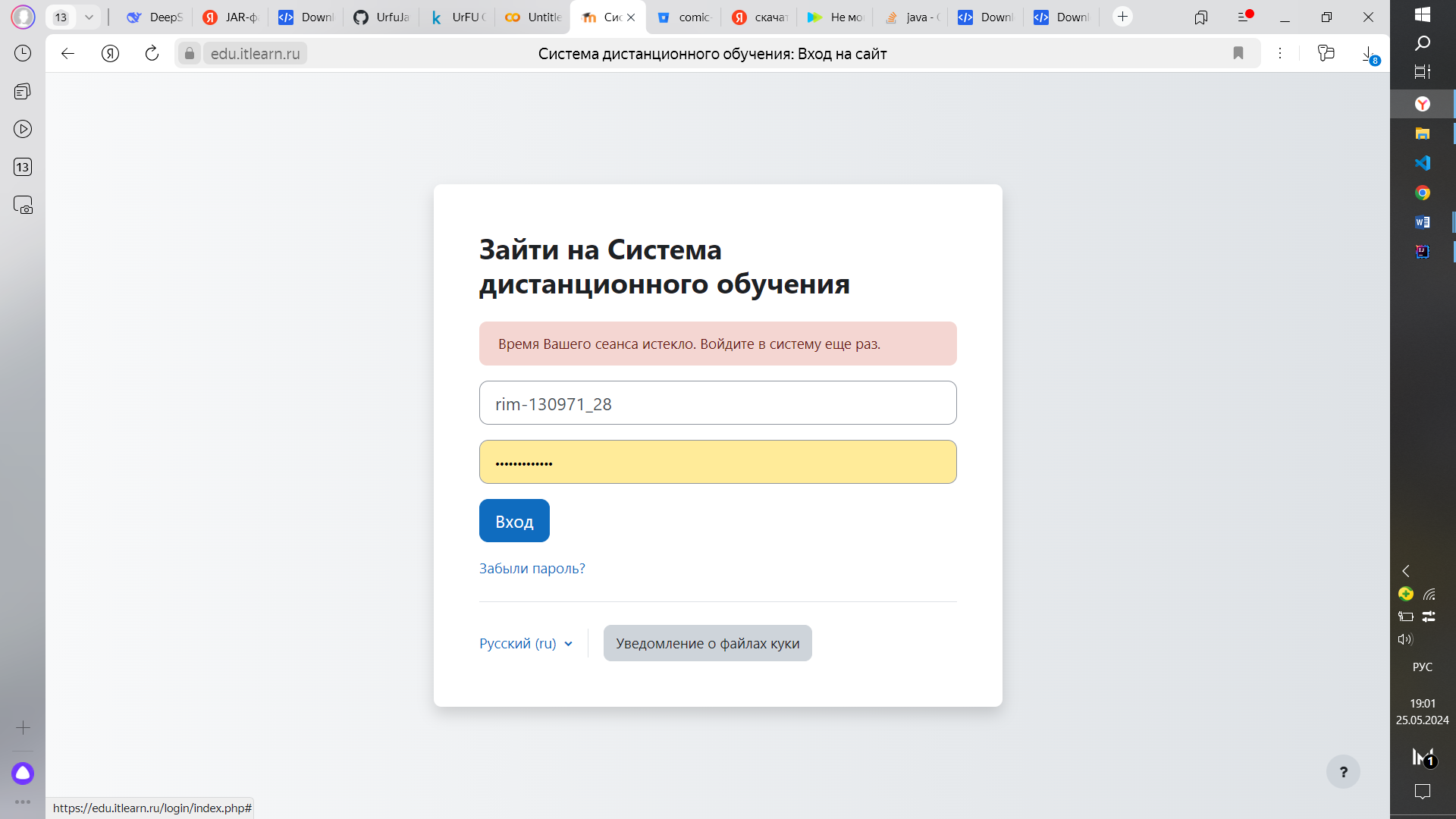
Код представляет собой простой пример Java-приложения для парсинга XML-документа. Этот класс использует библиотеку Java javax.xml.parsers для работы с XML.

* Инициализация файла: Создается объект File для указания пути к XML-файлу.
* Создание фабрики и построителя документа: Используется DocumentBuilderFactory для создания DocumentBuilder, который в свою очередь используется для парсинга XML-файла в объект Document.
* Парсинг XML: DocumentBuilder анализирует XML-файл и создает Document объект.
* Нормализация документа: doc.getDocumentElement().normalize(); используется для упрощения структуры документа и удаления лишних пустых текстовых узлов.
* Вывод корневого элемента: Выводится имя корневого элемента документа.
* Обработка элементов book: Получается список узлов с именем book и проходится в цикле. Для каждого элемента book:
* Выводится имя текущего элемента.
* Если элемент является элементом типа ELEMENT\_NODE (то есть не текстовым или другим типом узла), то извлекаются дочерние элементы title, author и year и выводятся их текстовые содержимое.



2. Добавьте возможность записывать новых сотрудников в XML-файл

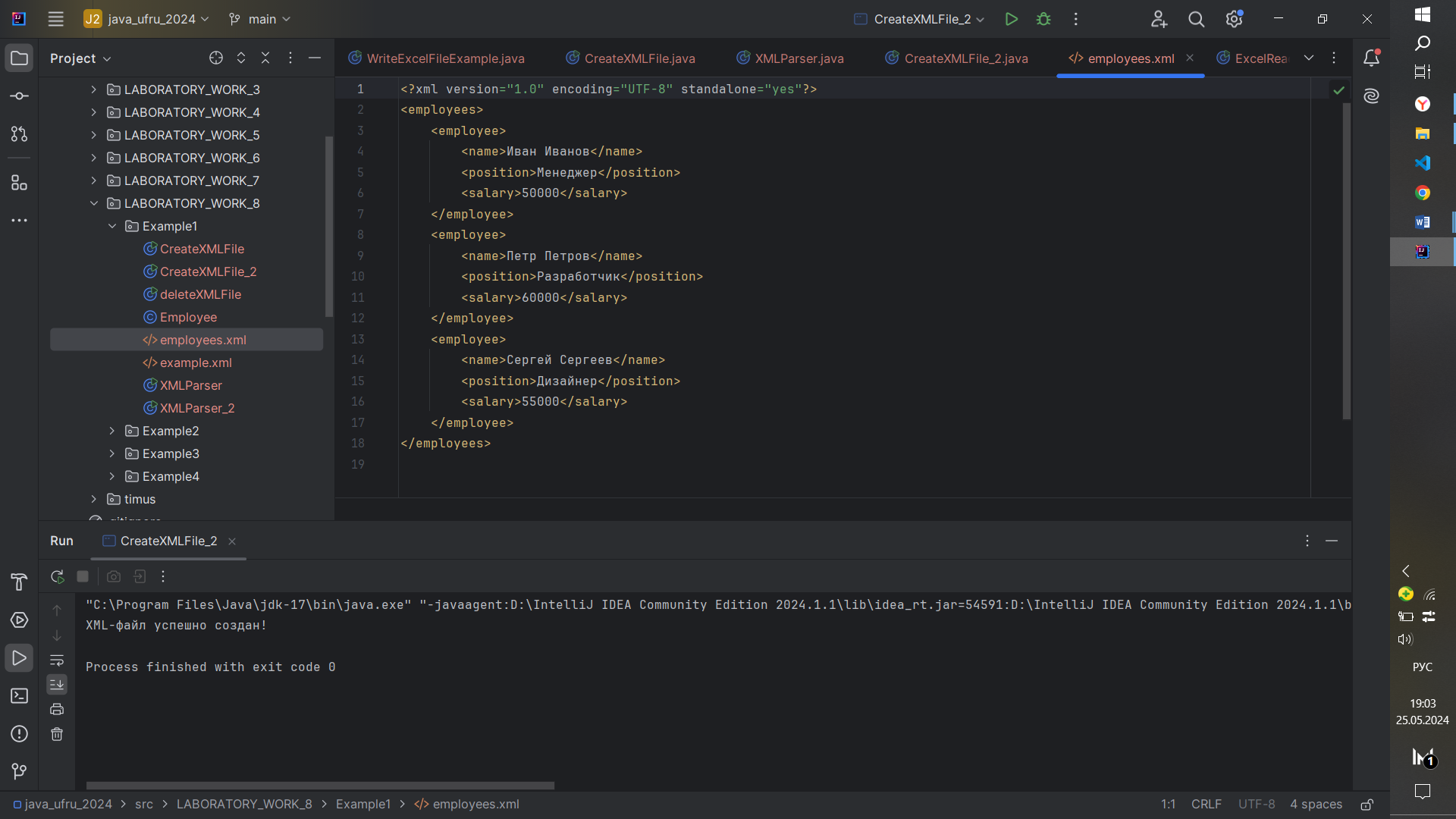
8. Список сотрудников (Employee);



public class CreateXMLFile\_2 {  
 public static void main(String[] args) {  
 try {  
 List<Employee> employees = new ArrayList<>();  
 employees.add(new Employee("Иван Иванов", "Менеджер", "50000"));  
 employees.add(new Employee("Петр Петров", "Разработчик", "60000"));  
 employees.add(new Employee("Сергей Сергеев", "Дизайнер", "55000"));  
  
 DocumentBuilderFactory docFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder docBuilder = docFactory.newDocumentBuilder();  
  
 Document doc = docBuilder.newDocument();  
 Element rootElement = doc.createElement("employees");  
 doc.appendChild(rootElement);  
  
 for (Employee employee : employees) {  
 *addEmployeeToXML*(doc, rootElement, employee);  
 }  
  
 doc.setXmlStandalone(true);  
 doc.normalizeDocument();  
 TransformerFactory tf = TransformerFactory.*newInstance*();  
 Transformer transformer = tf.newTransformer();  
 transformer.setOutputProperty(OutputKeys.*ENCODING*, "UTF-8");  
 transformer.setOutputProperty(OutputKeys.*STANDALONE*, "yes");  
 transformer.setOutputProperty(OutputKeys.*INDENT*, "yes");  
 DOMSource source = new DOMSource(doc);  
 StreamResult result = new StreamResult(new File("src/LABORATORY\_WORK\_8/Example1/employees.xml"));  
 transformer.transform(source, result);  
 System.*out*.println("XML-файл успешно создан!");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static void addEmployeeToXML(Document doc, Element rootElement, Employee employee) {  
 Element employeeElement = doc.createElement("employee");  
 rootElement.appendChild(employeeElement);  
  
 Element nameElement = doc.createElement("name");  
 nameElement.appendChild(doc.createTextNode(employee.name));  
 employeeElement.appendChild(nameElement);  
  
 Element positionElement = doc.createElement("position");  
 positionElement.appendChild(doc.createTextNode(employee.position));  
 employeeElement.appendChild(positionElement);  
  
 Element salaryElement = doc.createElement("salary");  
 salaryElement.appendChild(doc.createTextNode(employee.salary));  
 employeeElement.appendChild(salaryElement);  
 }  
}

Класс CreateXMLFile\_2 демонстрирует процесс создания XML-документа в Java с использованием DOM API. В этом примере создается XML-файл, содержащий информацию о сотрудниках.

* Инициализация списка сотрудников: Создается список employees, содержащий объекты класса Employee. Каждый объект Employee содержит имя, должность и зарплату сотрудника.
* Создание фабрики и построителя документа: Используется DocumentBuilderFactory для создания DocumentBuilder, который в свою очередь используется для создания нового Document.
* Создание корневого элемента: Создается элемент employees, который становится корневым элементом документа.
* Добавление сотрудников в XML: Для каждого сотрудника в списке вызывается метод addEmployeeToXML, который добавляет информацию о сотруднике в XML-документ.
* Настройка и сохранение XML: Устанавливается атрибут standalone документа в true, нормализуется документ. Затем создается Transformer для преобразования DOM-дерева в XML-файл. Устанавливаются параметры вывода, такие как кодировка, признак самостоятельности и отступы. XML-файл сохраняется в указанном пути.

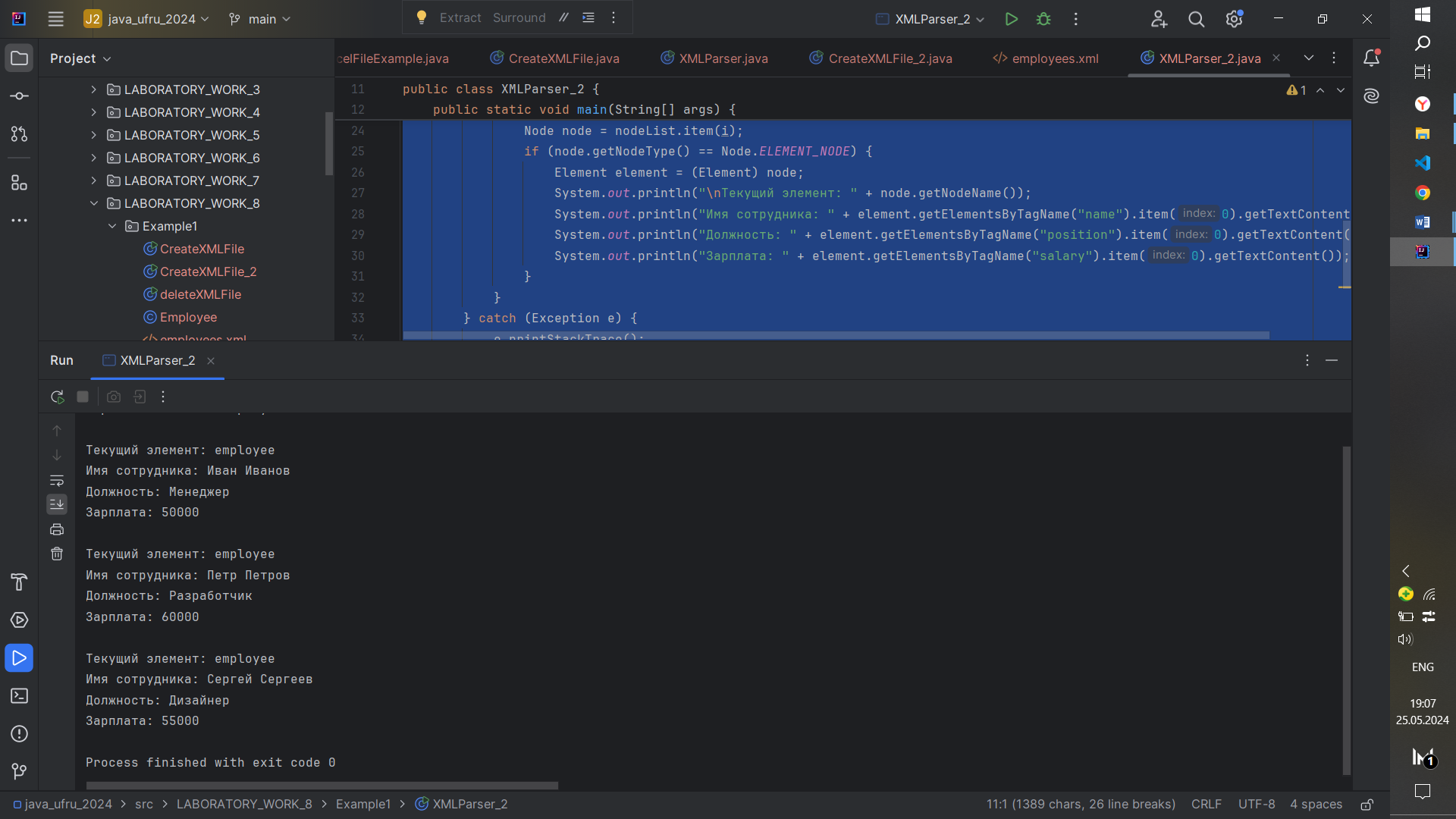


3. Добавьте возможность поиска по фамилий имени.

public class XMLParser\_2 {  
 public static void main(String[] args) {  
 try {  
 File inputFile = new File("src/LABORATORY\_WORK\_8/Example1/employees.xml");  
 DocumentBuilderFactory dbFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();  
 Document doc = dBuilder.parse(inputFile);  
 doc.getDocumentElement().normalize();  
  
 System.*out*.println("Корневой элемент: " + doc.getDocumentElement().getNodeName());  
 NodeList nodeList = doc.getElementsByTagName("employee");  
  
 for (int i = 0; i < nodeList.getLength(); i++) {  
 Node node = nodeList.item(i);  
 if (node.getNodeType() == Node.*ELEMENT\_NODE*) {  
 Element element = (Element) node;  
 System.*out*.println("\nТекущий элемент: " + node.getNodeName());  
 System.*out*.println("Имя сотрудника: " + element.getElementsByTagName("name").item(0).getTextContent());  
 System.*out*.println("Должность: " + element.getElementsByTagName("position").item(0).getTextContent());  
 System.*out*.println("Зарплата: " + element.getElementsByTagName("salary").item(0).getTextContent());  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Класс XMLParser\_2 представляет собой Java-приложение для парсинга XML-документа, содержащего информацию о сотрудниках. Этот класс использует библиотеку Java javax.xml.parsers для работы с XML.

* Инициализация файла: Создается объект File для указания пути к XML-файлу.
* Создание фабрики и построителя документа: Используется DocumentBuilderFactory для создания DocumentBuilder, который в свою очередь используется для парсинга XML-файла в объект Document.
* Парсинг XML: DocumentBuilder анализирует XML-файл и создает Document объект.
* Нормализация документа: doc.getDocumentElement().normalize(); используется для упрощения структуры документа и удаления лишних пустых текстовых узлов.
* Вывод корневого элемента: Выводится имя корневого элемента документа.
* Обработка элементов employee: Получается список узлов с именем employee и проходится в цикле. Для каждого элемента employee: Выводится имя текущего элемента. Если элемент является элементом типа ELEMENT\_NODE (то есть не текстовым или другим типом узла), то извлекаются дочерние элементы name, position и salary и выводятся их текстовые содержимое.

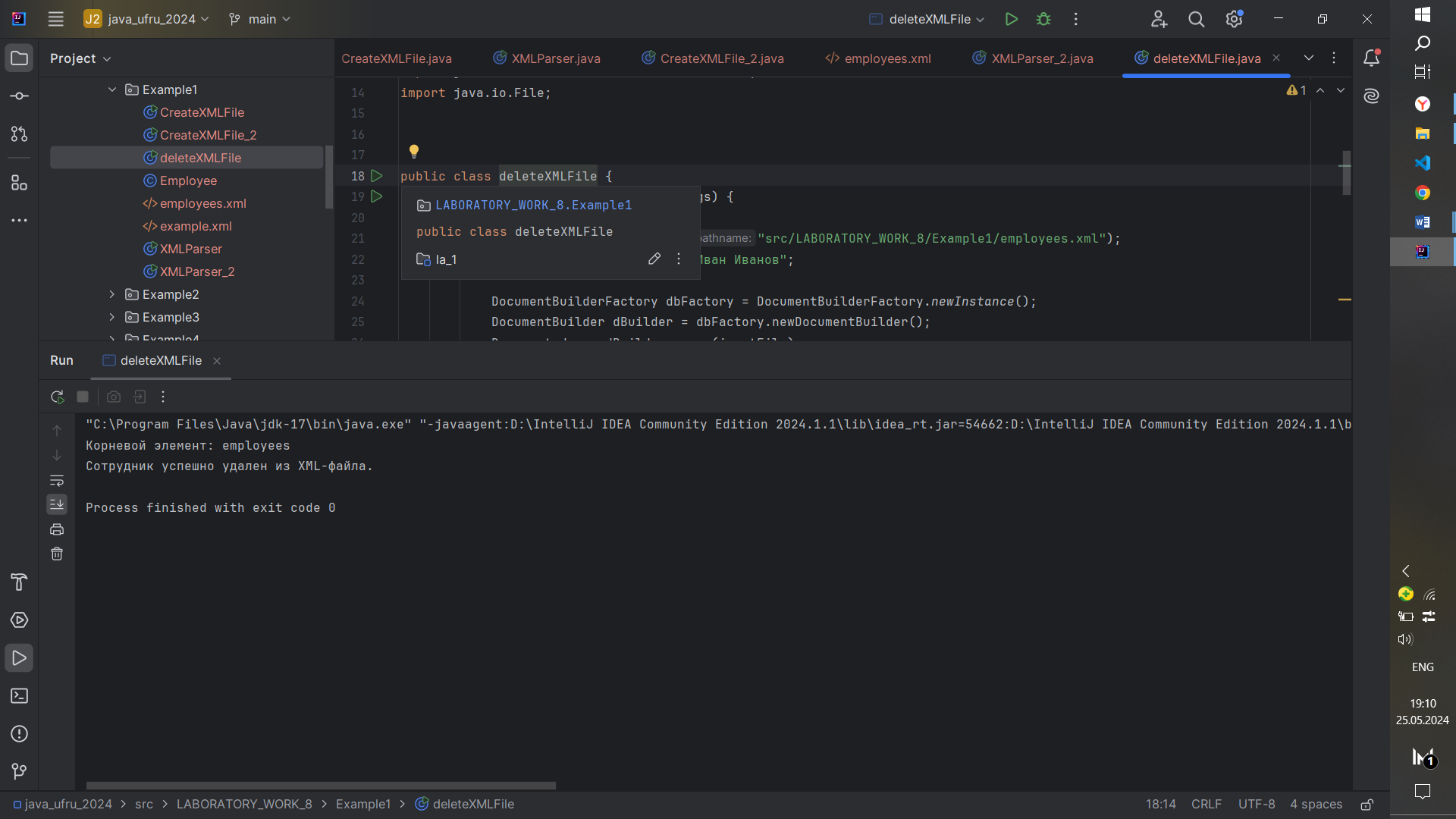


4. Реализуйте функцию удаления сотрудников из XML-файла.

public class deleteXMLFile {  
 public static void main(String[] args) {  
 try {  
 File inputFile = new File("src/LABORATORY\_WORK\_8/Example1/employees.xml");  
 String employeeToRemove = "Иван Иванов";  
  
 DocumentBuilderFactory dbFactory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();  
 Document doc = dBuilder.parse(inputFile);  
 doc.getDocumentElement().normalize();  
  
 System.*out*.println("Корневой элемент: " + doc.getDocumentElement().getNodeName());  
 NodeList nodeList = doc.getElementsByTagName("employee");  
  
 for (int i = 0; i < nodeList.getLength(); i++) {  
 Node node = nodeList.item(i);  
 if (node.getNodeType() == Node.*ELEMENT\_NODE*) {  
 Element element = (Element) node;  
 String name = element.getElementsByTagName("name").item(0).getTextContent();  
 if (name.equals(employeeToRemove)) {  
 node.getParentNode().removeChild(node);  
 break;  
 }  
 }  
 }  
  
 TransformerFactory transformerFactory = TransformerFactory.*newInstance*();  
 Transformer transformer = transformerFactory.newTransformer();  
 transformer.setOutputProperty(OutputKeys.*INDENT*, "yes");  
 transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");  
 DOMSource source = new DOMSource(doc);  
 StreamResult result = new StreamResult(inputFile);  
 transformer.transform(source, result);  
  
 System.*out*.println("Сотрудник успешно удален из XML-файла.");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
}

Класс deleteXMLFile демонстрирует процесс удаления элемента из XML-документа в Java с использованием DOM API. В этом примере из XML-файла удаляется сотрудник по имени.

* Инициализация файла и имени сотрудника для удаления: Создается объект File для указания пути к XML-файлу, а также переменная employeeToRemove для хранения имени сотрудника, которого нужно удалить.
* Создание фабрики и построителя документа: Используется DocumentBuilderFactory для создания DocumentBuilder, который в свою очередь используется для парсинга XML-файла в объект Document.
* Парсинг XML: DocumentBuilder анализирует XML-файл и создает Document объект.
* Нормализация документа: doc.getDocumentElement().normalize(); используется для упрощения структуры документа и удаления лишних пустых текстовых узлов.
* Поиск и удаление сотрудника: Получается список узлов с именем employee и проходится в цикле. Для каждого элемента employee: Извлекается имя сотрудника. Если имя совпадает с именем, которое нужно удалить, то элемент employee удаляется из своего родительского узла.
* Сохранение изменений в XML: Создается Transformer для преобразования DOM-дерева в XML-файл с отступами для улучшения читаемости. XML-файл сохраняется в исходном файле.

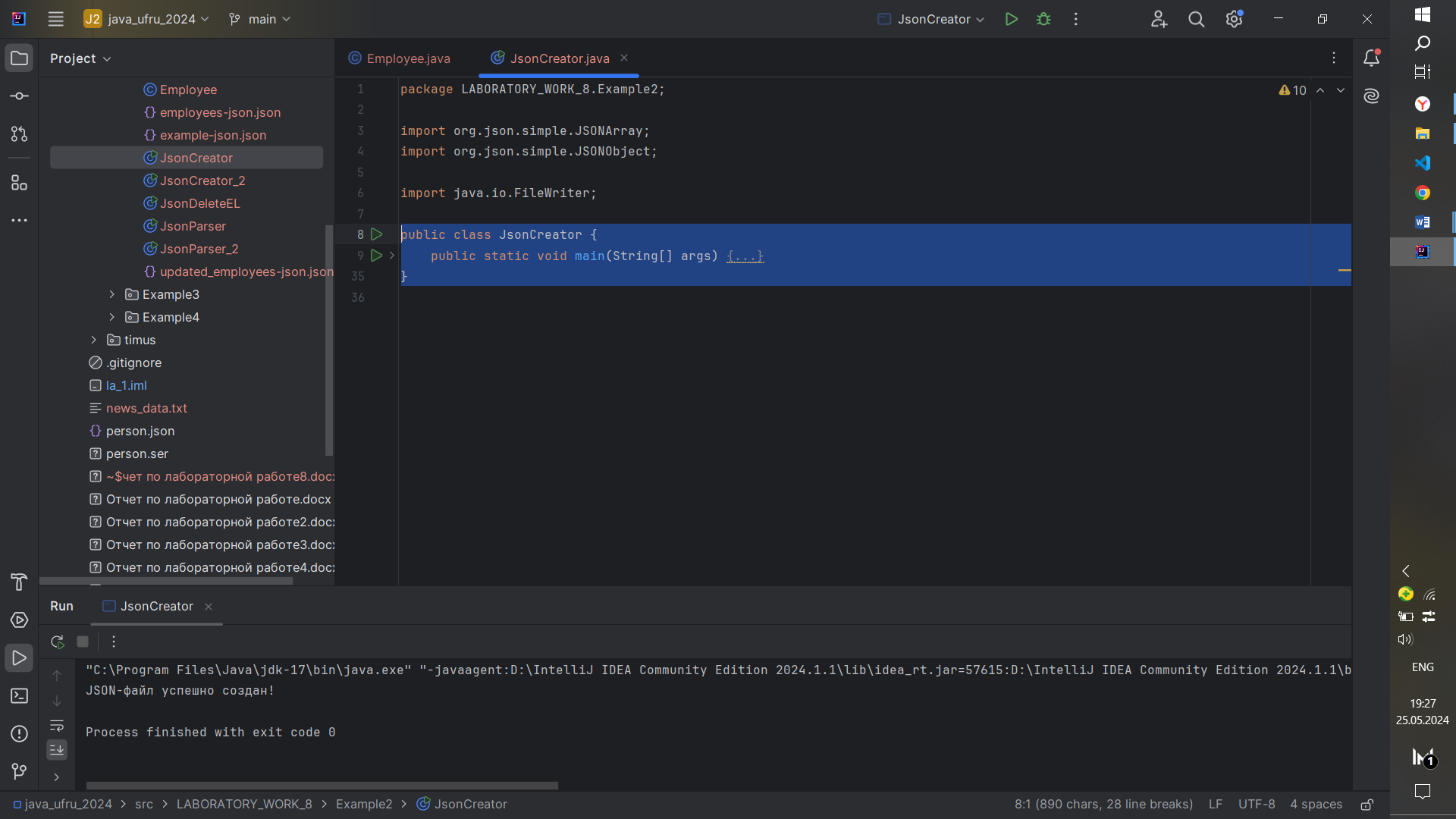


Example2

public class JsonCreator {  
 public static void main(String[] args) {  
 JSONObject library = new JSONObject();  
 JSONArray books = new JSONArray();  
  
 JSONObject book1 = new JSONObject();  
 book1.put("title", "Война и мир");  
 book1.put("author", "Лев Толстой");  
 book1.put("year", 1869);  
  
 JSONObject book2 = new JSONObject();  
 book2.put("title", "Мастер и Маргарита");  
 book2.put("author", "Михаил Булгаков");  
 book2.put("year", 1967);  
  
 books.add(book1);  
 books.add(book2);  
  
 library.put("books", books);  
  
 try (FileWriter file = new FileWriter("src/LABORATORY\_WORK\_8/Example2/example-json.json")) {  
 file.write(library.toJSONString());  
 System.*out*.println("JSON-файл успешно создан!");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Класс JsonCreator демонстрирует процесс создания JSON-документа в Java с использованием библиотеки org.json. В этом примере создается JSON-файл, содержащий информацию о книгах в библиотеке.

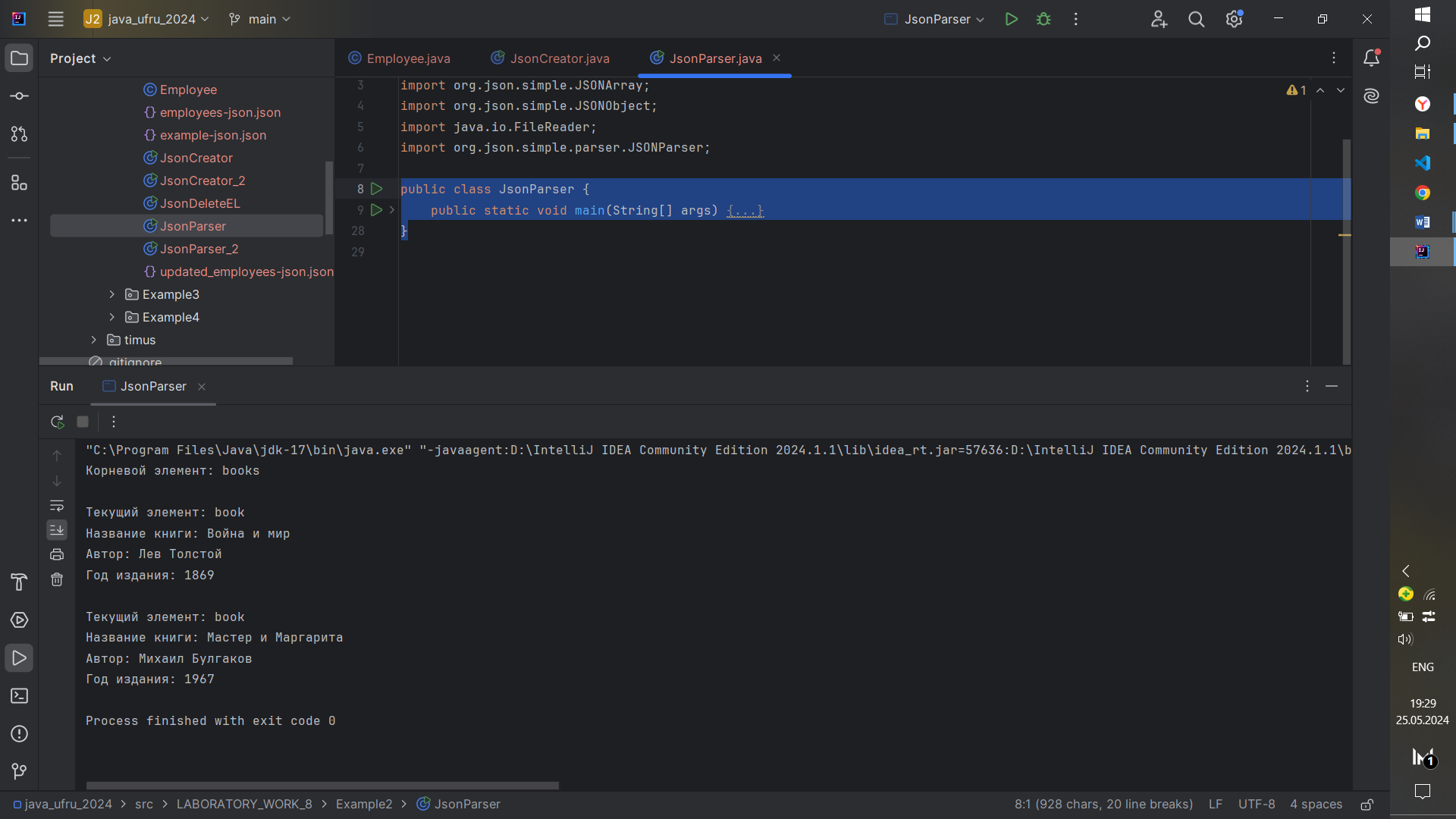
* Создание JSON-объекта библиотеки: Создается объект JSONObject для представления библиотеки.
* Создание массива книг: Создается объект JSONArray для хранения информации о книгах.
* Добавление книг в массив: Создаются два объекта JSONObject для каждой книги, содержащие поля title, author и year. Эти объекты добавляются в массив books.
* Добавление массива книг в библиотеку: Массив books добавляется в объект library с ключом books.
* Запись JSON в файл: Используется FileWriter для записи JSON-строки, представленной объектом library, в файл. Устанавливается режим автоматического закрытия файла с помощью оператора try-with-resources.



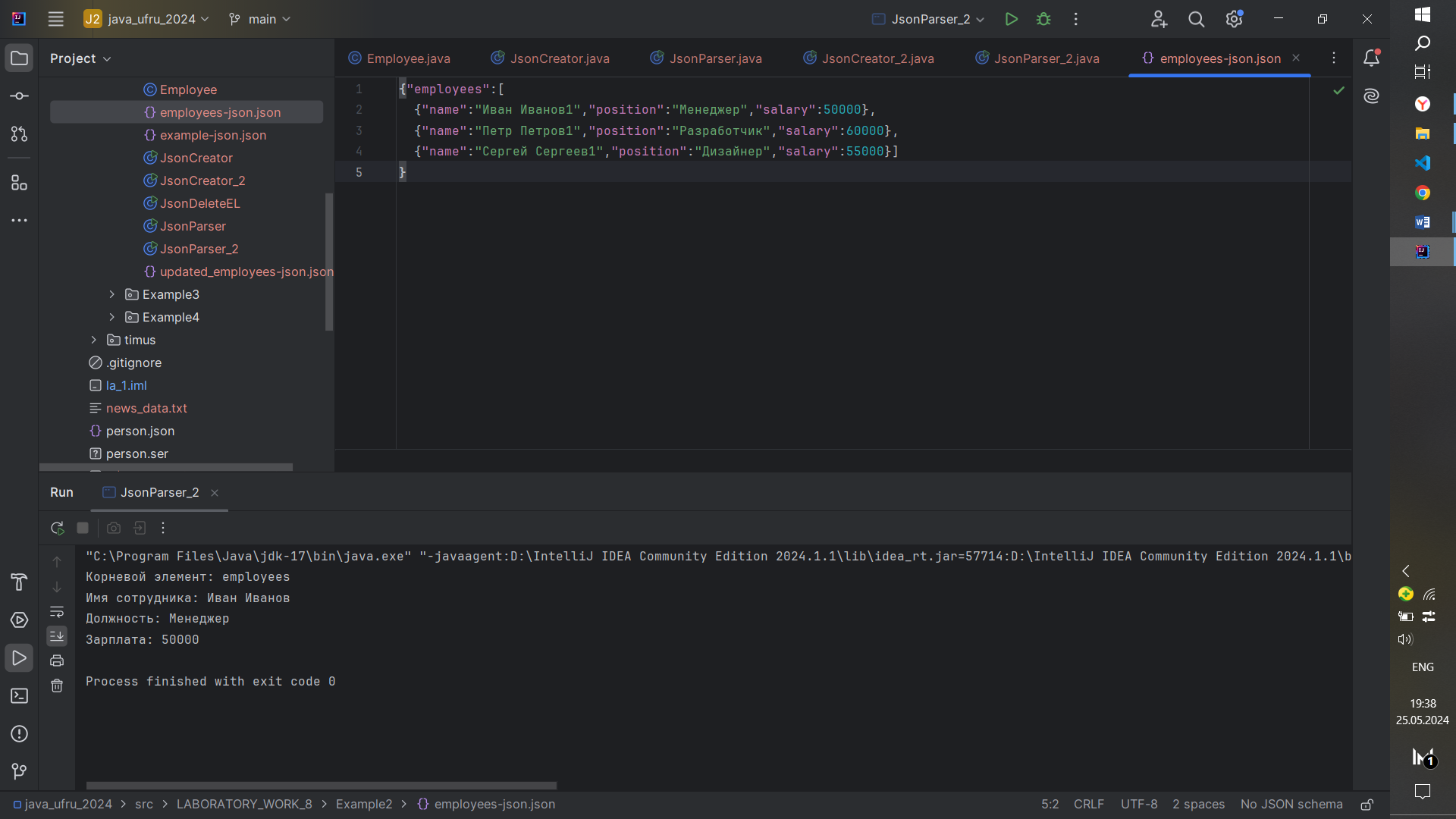
public class JsonParser {  
 public static void main(String[] args) {  
 try {  
 JSONParser parser = new JSONParser();  
 Object obj = parser.parse(new FileReader("src/LABORATORY\_WORK\_8/Example2/example-json.json"));  
 JSONObject jsonObject = (JSONObject) obj;  
 System.*out*.println("Корневой элемент: " + jsonObject.keySet().iterator().next());  
 JSONArray jsonArray = (JSONArray) jsonObject.get("books");  
  
 for (Object o : jsonArray) {  
 JSONObject book = (JSONObject) o;  
 System.*out*.println("\nТекущий элемент: book");  
 System.*out*.println("Название книги: " + book.get("title"));  
 System.*out*.println("Автор: " + book.get("author"));  
 System.*out*.println("Год издания: " + book.get("year"));  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Класс JsonParser демонстрирует процесс парсинга JSON-документа в Java с использованием библиотеки org.json. В этом примере из JSON-файла извлекается информация о книгах.

* Инициализация парсера и чтение JSON: Создается объект JSONParser, который используется для чтения JSON-файла через FileReader.
* Парсинг JSON: JSONParser анализирует JSON-файл и создает объект JSONObject.
* Вывод корневого элемента: Выводится ключ корневого элемента JSON-объекта.
* Обработка массива книг: Из JSON-объекта извлекается массив books и проходится в цикле. Для каждого элемента массива:
* Выводится имя текущего элемента (в данном случае "book").
* Извлекаются и выводятся значения полей title, author и year каждой книги.



1.

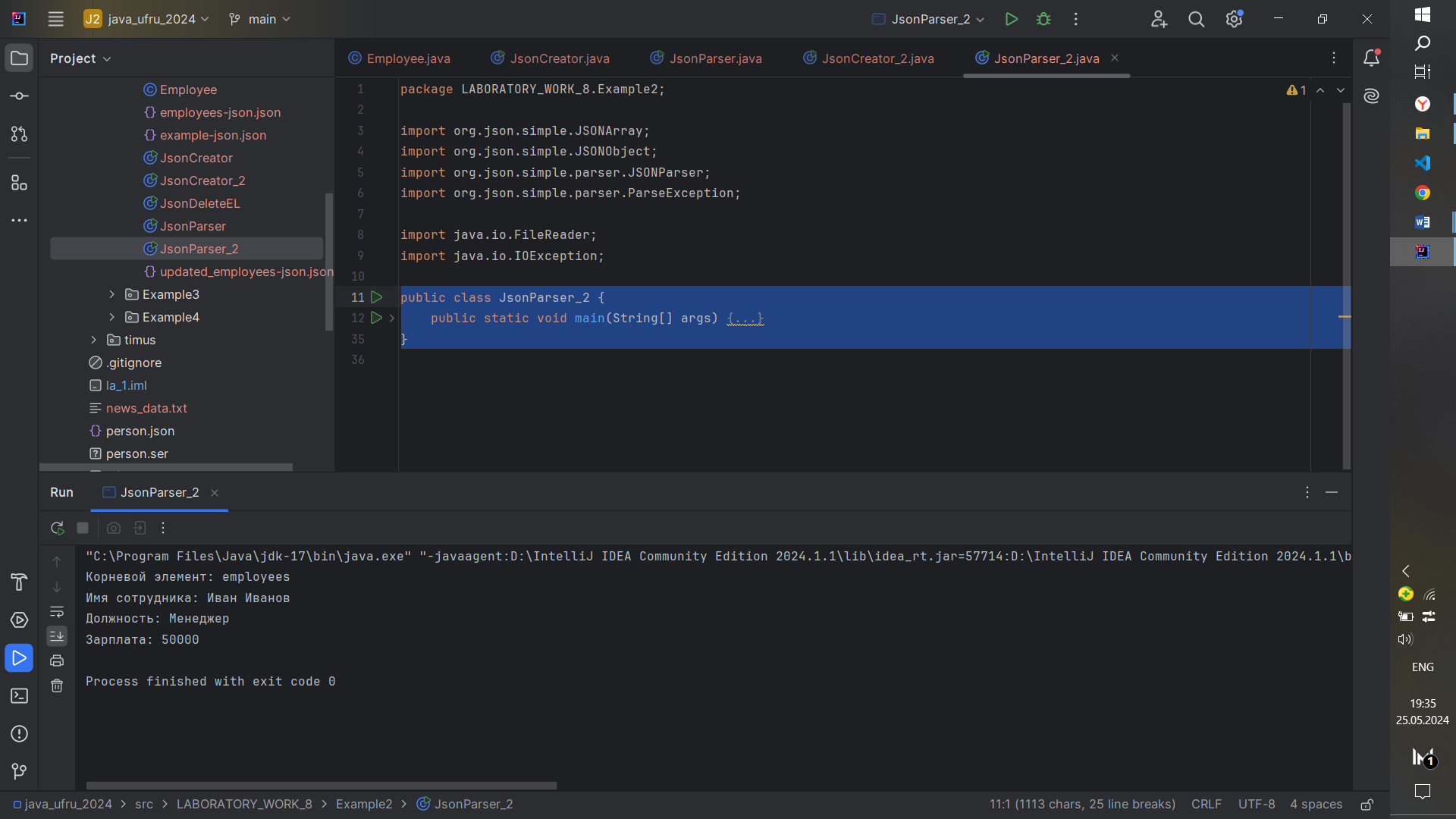


2. Добавьте функционал для поиска сотрудников по фамилий.

public class JsonParser\_2 {  
 public static void main(String[] args) {  
 try {  
 JSONParser parser = new JSONParser();  
 Object obj = parser.parse(new FileReader("src/LABORATORY\_WORK\_8/Example2/employees-json.json"));  
 JSONObject jsonObject = (JSONObject) obj;  
 System.*out*.println("Корневой элемент: " + jsonObject.keySet().iterator().next());  
 JSONArray jsonArray = (JSONArray) jsonObject.get("employees");  
  
 String searchLastName = "Иван Иванов"; // Фамилия для поиска  
  
 for (Object o : jsonArray) {  
 JSONObject employee = (JSONObject) o;  
 String name = (String) employee.get("name");  
 if (name.equals(searchLastName)) {  
 System.*out*.println("Имя сотрудника: " + name);  
 System.*out*.println("Должность: " + employee.get("position"));  
 System.*out*.println("Зарплата: " + employee.get("salary"));  
 }  
 }  
 } catch (IOException | ParseException e) {  
 e.printStackTrace();  
 }  
 }  
}

Класс JsonParser\_2 демонстрирует процесс парсинга JSON-документа в Java с использованием библиотеки org.json. В этом примере из JSON-файла извлекается информация о сотрудниках, и выполняется поиск сотрудника по фамилии.

* Инициализация парсера и чтение JSON: Создается объект JSONParser, который используется для чтения JSON-файла через FileReader.
* Парсинг JSON: JSONParser анализирует JSON-файл и создает объект JSONObject.
* Вывод корневого элемента: Выводится ключ корневого элемента JSON-объекта.
* Обработка массива сотрудников: Из JSON-объекта извлекается массив employees и проходится в цикле. Для каждого элемента массива: Извлекается имя сотрудника. Если имя совпадает с указанной фамилией для поиска, то извлекаются и выводятся должность и зарплата сотрудника.

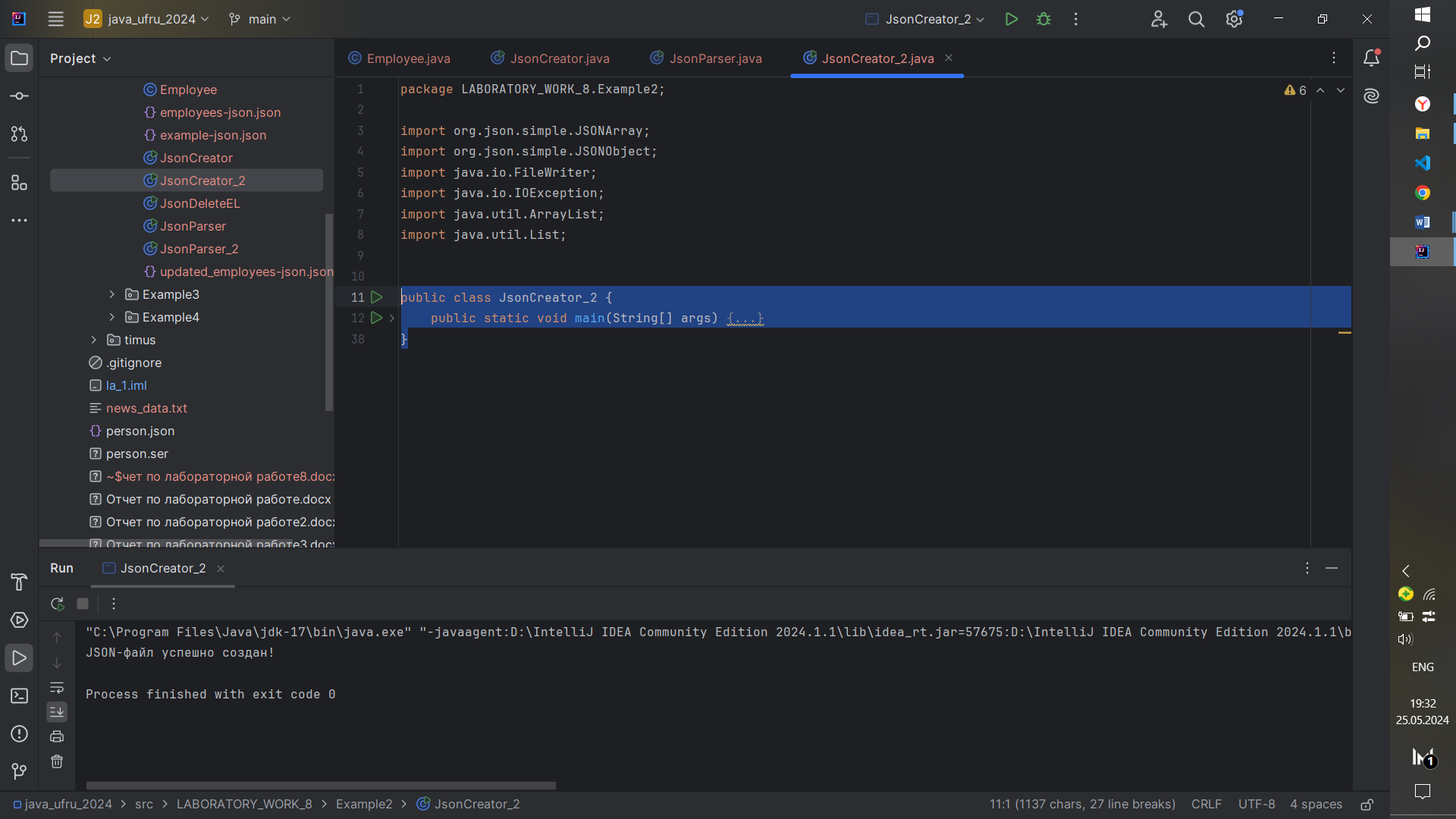


3. Добавьте функционал для добавления новых сотрудников.

public class JsonCreator\_2 {  
 public static void main(String[] args) {  
 List<Employee> employees = new ArrayList<>();  
 employees.add(new Employee("Иван Иванов", "Менеджер", 50000));  
 employees.add(new Employee("Петр Петров", "Разработчик", 60000));  
 employees.add(new Employee("Сергей Сергеев", "Дизайнер", 55000));  
  
 JSONObject employeesJson = new JSONObject();  
 JSONArray employeeList = new JSONArray();  
  
 for (Employee employee : employees) {  
 JSONObject employeeJson = new JSONObject();  
 employeeJson.put("name", employee.name);  
 employeeJson.put("position", employee.position);  
 employeeJson.put("salary", employee.salary);  
 employeeList.add(employeeJson);  
 }  
  
 employeesJson.put("employees", employeeList);  
  
 try (FileWriter file = new FileWriter("src/LABORATORY\_WORK\_8/Example2/employees-json.json")) {  
 file.write(employeesJson.toJSONString());  
 System.*out*.println("JSON-файл успешно создан!");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

Класс JsonCreator\_2 демонстрирует процесс создания JSON-документа в Java с использованием библиотеки org.json. В этом примере создается JSON-файл, содержащий информацию о сотрудниках.

* Инициализация списка сотрудников: Создается список employees, содержащий объекты класса Employee. Каждый объект Employee содержит имя, должность и зарплату сотрудника.
* Создание JSON-объекта и массива: Создаются объекты JSONObject для представления всех сотрудников и JSONArray для хранения информации о каждом сотруднике.
* Добавление сотрудников в JSON: Для каждого сотрудника в списке создается объект JSONObject, который содержит поля name, position и salary. Эти объекты добавляются в массив employeeList.
* Добавление массива сотрудников в JSON-объект: Массив employeeList добавляется в объект employeesJson с ключом employees.
* Запись JSON в файл: Используется FileWriter для записи JSON-строки, представленной объектом employeesJson, в файл. Устанавливается режим автоматического закрытия файла с помощью оператора try-with-resources.

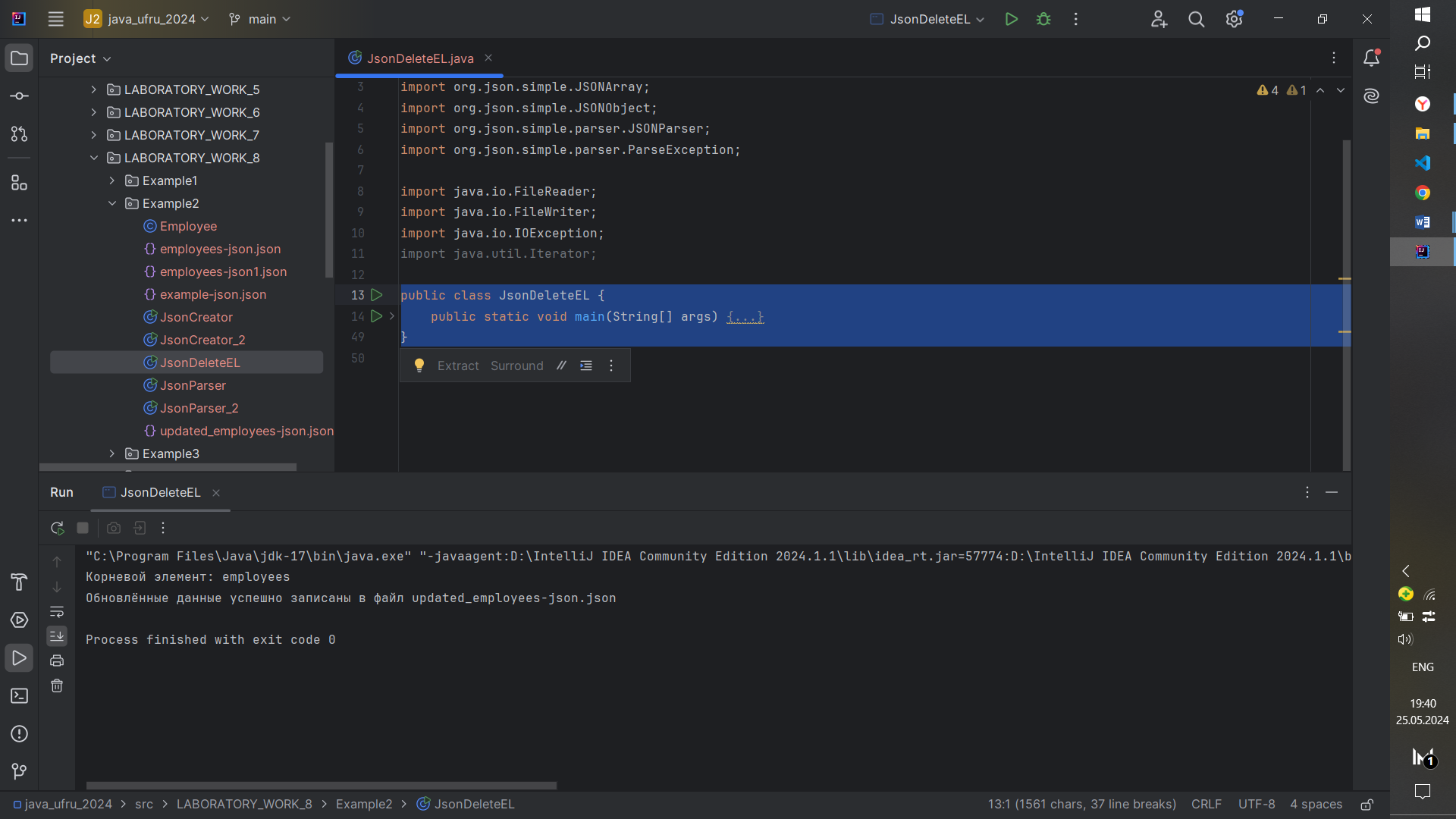


4. Добавьте функционал для удаления.

public class JsonDeleteEL {  
 public static void main(String[] args) {  
 try {  
 JSONParser parser = new JSONParser();  
 Object obj = parser.parse(new FileReader("src/LABORATORY\_WORK\_8/Example2/employees-json.json"));  
 JSONObject jsonObject = (JSONObject) obj;  
 System.*out*.println("Корневой элемент: " + jsonObject.keySet().iterator().next());  
 JSONArray jsonArray = (JSONArray) jsonObject.get("employees");  
  
 String searchLastName = "Иван Иванов"; // Фамилия для поиска и удаления  
  
 // Создание нового JSONArray для обновлённых данных  
 JSONArray updatedArray = new JSONArray();  
  
 // Итерация по массиву сотрудников  
 for (Object o : jsonArray) {  
 JSONObject employee = (JSONObject) o;  
 String name = (String) employee.get("name");  
 if (!name.equals(searchLastName)) {  
 updatedArray.add(employee);  
 }  
 }  
  
 // Обновление корневого объекта  
 jsonObject.put("employees", updatedArray);  
  
 // Запись обновлённых данных в новый файл  
 try (FileWriter file = new FileWriter("src/LABORATORY\_WORK\_8/Example2/updated\_employees-json.json")) {  
 file.write(jsonObject.toJSONString());  
 System.*out*.println("Обновлённые данные успешно записаны в файл updated\_employees-json.json");  
 }  
  
 } catch (IOException | ParseException e) {  
 e.printStackTrace();  
 }  
 }  
}

Класс JsonDeleteEL демонстрирует процесс удаления элемента из JSON-документа в Java с использованием библиотеки org.json. В этом примере из JSON-файла удаляется сотрудник по фамилии.

* Инициализация парсера и чтение JSON: Создается объект JSONParser, который используется для чтения JSON-файла через FileReader.
* Парсинг JSON: JSONParser анализирует JSON-файл и создает объект JSONObject.
* Вывод корневого элемента: Выводится ключ корневого элемента JSON-объекта.
* Обработка массива сотрудников: Из JSON-объекта извлекается массив employees и проходится в цикле. Для каждого элемента массива: Извлекается имя сотрудника. Если имя не совпадает с указанной фамилией для удаления, то объект сотрудника добавляется в новый массив updatedArray.
* Обновление корневого объекта: Массив updatedArray заменяет исходный массив employees в корневом объекте jsonObject.
* Запись обновлённых данных в новый файл: Используется FileWriter для записи JSON-строки, представленной объектом jsonObject, в новый файл. Устанавливается режим автоматического закрытия файла с помощью оператора try-with-resources.

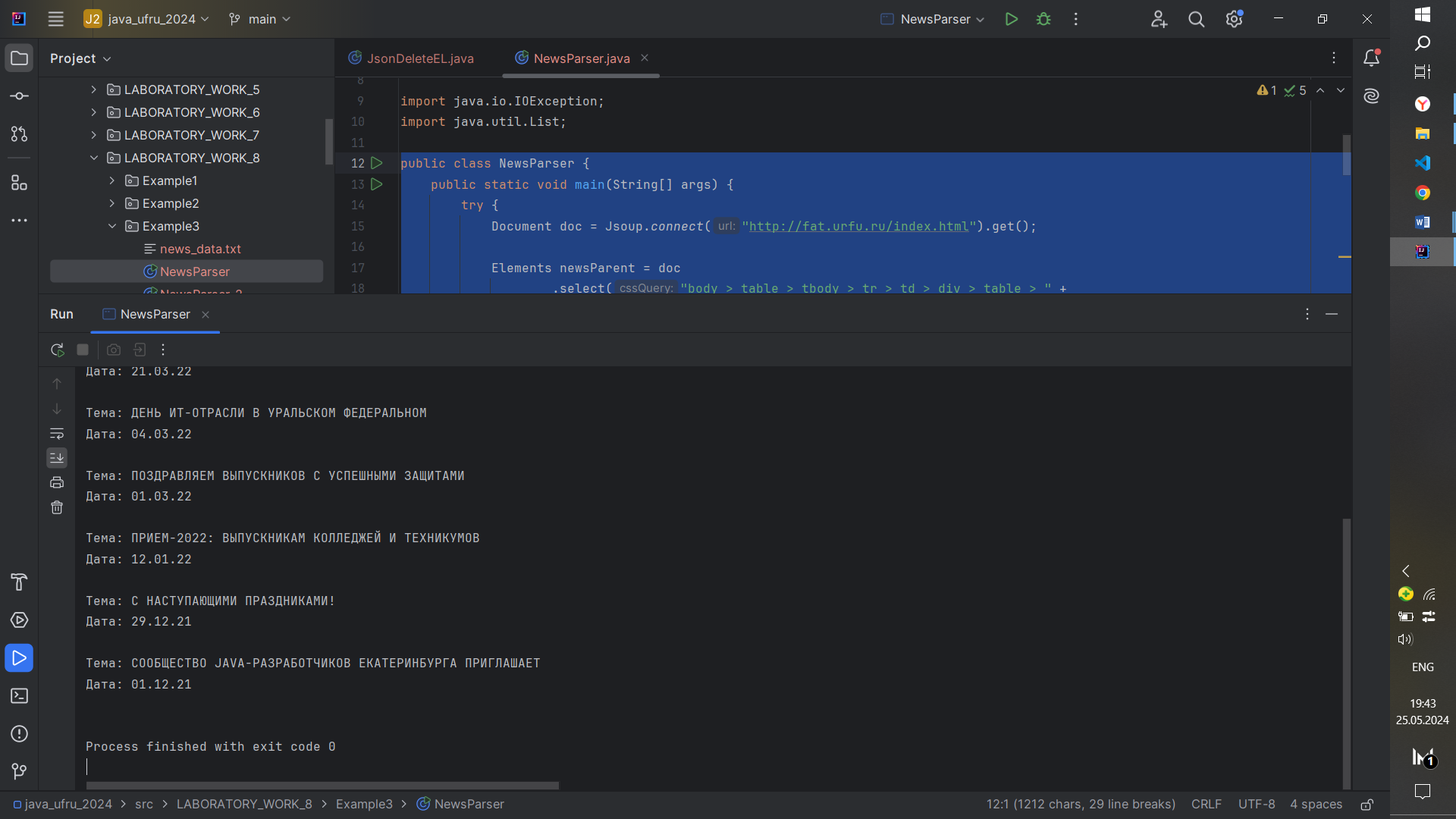


Example3

public class NewsParser {  
 public static void main(String[] args) {  
 try {  
 Document doc = Jsoup.*connect*("http://fat.urfu.ru/index.html").get();  
  
 Elements newsParent = doc  
 .select("body > table > tbody > tr > td > div > table > " +  
 "tbody > tr:nth-child(5) > td:nth-child(3) > table > tbody > " +  
 "tr > td:nth-child(1)");  
 for (int i = 3; i < 20; i++) {  
 if (!(i % 2 == 0)) {  
 List<Node> nodes = newsParent.get(0).childNodes();  
 System.*out*.println("Тема: " +  
 ((Element) nodes.get(i))  
 .getElementsByClass("blocktitle")  
 .get(0).childNodes().get(0));  
  
 System.*out*.println("Дата: " +  
 ((Element) nodes.get(i))  
 .getElementsByClass("blockdate")  
 .get(0).childNodes().get(0) +  
 "\n");  
  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

Класс NewsParser демонстрирует процесс парсинга новостного сайта с использованием библиотеки jsoup для работы с HTML. В этом примере из веб-страницы извлекается информация о новостях, включая темы и даты.

* Подключение к веб-странице и получение HTML: Используется Jsoup.connect("http://fat.urfu.ru/index.html").get() для подключения к указанному URL и получения HTML-документа.
* Выбор элементов новостей: Используется метод select для выбора элементов HTML, содержащих новости. Выражение CSS body > table > tbody > tr > td > div > table > tbody > tr:nth-child(5) > td:nth-child(3) > table > tbody > tr > td:nth-child(1) указывает на родительский элемент, содержащий новости.
* Парсинг новостей: Проходится цикл от 3 до 19, игнорируя четные индексы (предполагается, что новости расположены в нечетных индексах). Для каждого элемента:
* Извлекаются дочерние узлы родительского элемента.
* Извлекается тема новости: выбирается элемент с классом blocktitle и извлекается первый дочерний узел.
* Извлекается дата новости: выбирается элемент с классом blockdate и извлекается первый дочерний узел.

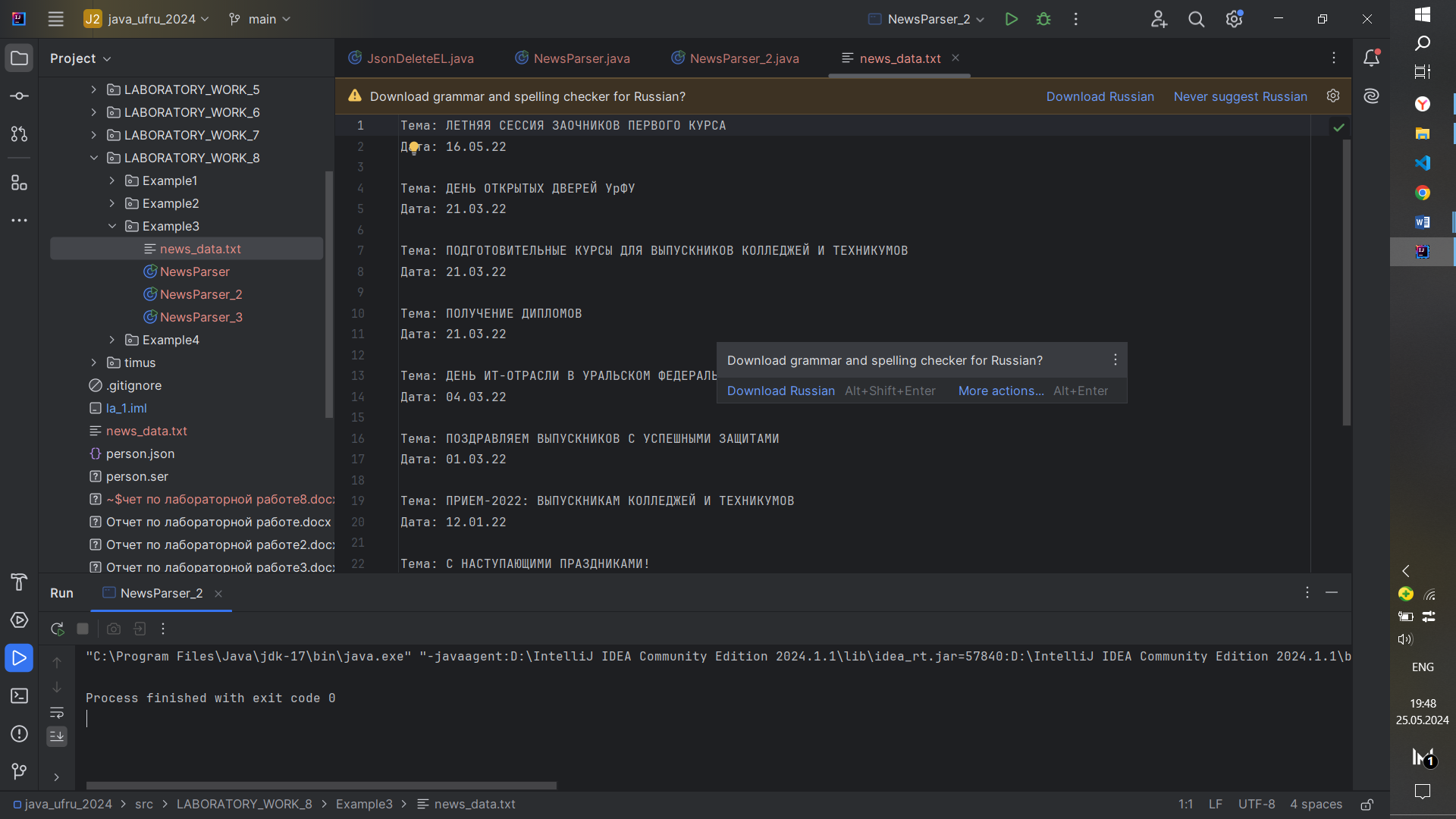


1. Добавьте функционал записи полученных данных в файл для сохранения информации на будущее.

public class NewsParser\_2 {  
 public static void main(String[] args) {  
 try {  
 Document doc = Jsoup.*connect*("http://fat.urfu.ru/index.html").get();  
  
 Elements newsParent = doc  
 .select("body > table > tbody > tr > td > div > table > " +  
 "tbody > tr:nth-child(5) > td:nth-child(3) > table > tbody > " +  
 "tr > td:nth-child(1)");  
  
 // Путь к файлу, в который будут записываться данные  
 String filePath = "src/LABORATORY\_WORK\_8/Example3/news\_data.txt";  
  
 // Используем try-with-resources для автоматического закрытия FileWriter  
 try (FileWriter writer = new FileWriter(filePath)) {  
 for (int i = 3; i < 20; i++) {  
 if (!(i % 2 == 0)) {  
 List<Node> nodes = newsParent.get(0).childNodes();  
 Element newsItem = (Element) nodes.get(i);  
  
 // Запись темы новости  
 writer.write("Тема: " +  
 newsItem.getElementsByClass("blocktitle").get(0).childNodes().get(0) + "\n");  
  
 // Запись даты новости  
 writer.write("Дата: " +  
 newsItem.getElementsByClass("blockdate").get(0).childNodes().get(0) + "\n\n");  
 }  
 }  
 }  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

Класс NewsParser\_2 демонстрирует процесс парсинга новостного сайта с использованием библиотеки jsoup для работы с HTML и записью извлеченных данных в текстовый файл. В этом примере из веб-страницы извлекается информация о новостях, включая темы и даты, и сохраняется в файл. Вот подробное описание каждой части кода:

* Подключение к веб-странице и получение HTML: Используется Jsoup.connect("http://fat.urfu.ru/index.html").get() для подключения к указанному URL и получения HTML-документа.
* Выбор элементов новостей: Используется метод select для выбора элементов HTML, содержащих новости. Выражение CSS указывает на родительский элемент, содержащий новости.
* Запись данных в файл: Проходится цикл от 3 до 19, игнорируя четные индексы (предполагается, что новости расположены в нечетных индексах). Для каждого элемента:
  + Извлекаются дочерние узлы родительского элемента.
  + Извлекается тема новости: выбирается элемент с классом blocktitle и извлекается первый дочерний узел.
  + Извлекается дата новости: выбирается элемент с классом blockdate и извлекается первый дочерний узел.
* Данные записываются в файл с использованием FileWriter. Устанавливается режим автоматического закрытия файла с помощью оператора try-with-resources.

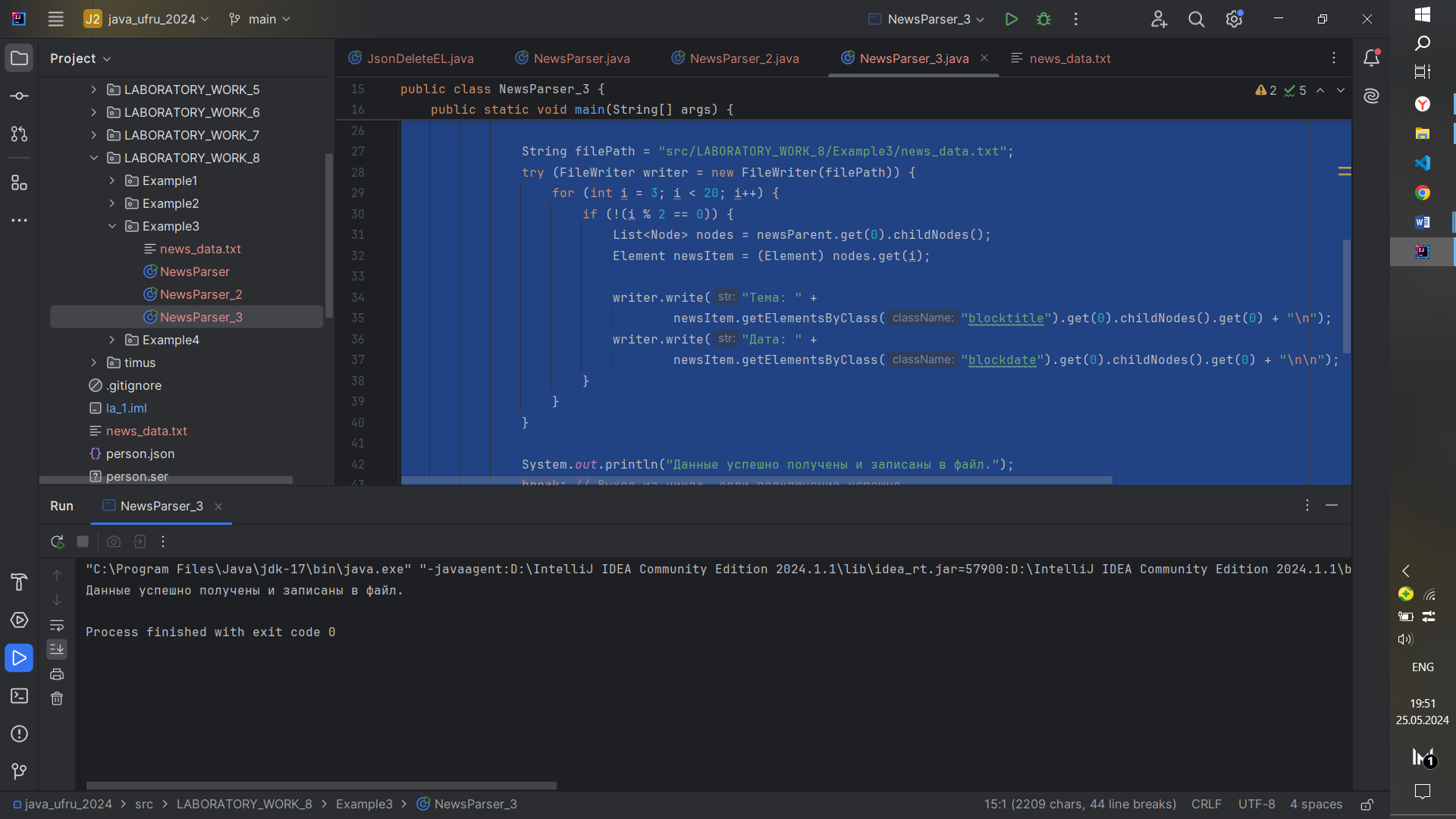


2. Добавьте обработку ошибок при получении HTML-кода страницы, например вывод сообщения об ошибке и попытка переподключения к сайту.

public class NewsParser\_3 {  
 public static void main(String[] args) {  
 int maxAttempts = 5; // Максимальное количество попыток подключения  
 for (int attempt = 1; attempt <= maxAttempts; attempt++) {  
 try {  
 Document doc = Jsoup.*connect*("http://fat.urfu.ru/index.html").get();  
  
 Elements newsParent = doc  
 .select("body > table > tbody > tr > td > div > table > " +  
 "tbody > tr:nth-child(5) > td:nth-child(3) > table > tbody > " +  
 "tr > td:nth-child(1)");  
  
 String filePath = "src/LABORATORY\_WORK\_8/Example3/news\_data.txt";  
 try (FileWriter writer = new FileWriter(filePath)) {  
 for (int i = 3; i < 20; i++) {  
 if (!(i % 2 == 0)) {  
 List<Node> nodes = newsParent.get(0).childNodes();  
 Element newsItem = (Element) nodes.get(i);  
  
 writer.write("Тема: " +  
 newsItem.getElementsByClass("blocktitle").get(0).childNodes().get(0) + "\n");  
 writer.write("Дата: " +  
 newsItem.getElementsByClass("blockdate").get(0).childNodes().get(0) + "\n\n");  
 }  
 }  
 }  
  
 System.*out*.println("Данные успешно получены и записаны в файл.");  
 break; // Выход из цикла, если подключение успешно  
  
 } catch (IOException e) {  
 if (attempt < maxAttempts) {  
 System.*out*.println("Не удалось подключиться к сайту. Попытка " + attempt + " из " + maxAttempts + ". Повторная попытка через 5 секунд...");  
 try {  
 Thread.*sleep*(5000); // Пауза между попытками  
 } catch (InterruptedException ex) {  
 Thread.*currentThread*().interrupt();  
 }  
 } else {  
 System.*out*.println("Не удалось подключиться к сайту после " + maxAttempts + " попыток.");  
 }  
 }  
 }  
 }  
}

Класс NewsParser\_3 демонстрирует процесс парсинга новостного сайта с использованием библиотеки jsoup для работы с HTML, включая повторные попытки подключения в случае неудачи и запись извлеченных данных в текстовый файл. В этом примере из веб-страницы извлекается информация о новостях, включая темы и даты, и сохраняется в файл с учетом возможных проблем с подключением.

* Повторные попытки подключения: Используется цикл, который позволяет повторить попытку подключения до 5 раз в случае неудачи.
* Подключение к веб-странице и получение HTML: Используется Jsoup.connect("http://fat.urfu.ru/index.html").get() для подключения к указанному URL и получения HTML-документа.
* Выбор элементов новостей: Используется метод select для выбора элементов HTML, содержащих новости. Выражение CSS указывает на родительский элемент, содержащий новости.
* Запись данных в файл: Проходится цикл от 3 до 19, игнорируя четные индексы (предполагается, что новости расположены в нечетных индексах). Для каждого элемента:
* Извлекаются дочерние узлы родительского элемента.
* Извлекается тема новости: выбирается элемент с классом blocktitle и извлекается первый дочерний узел.
* Извлекается дата новости: выбирается элемент с классом blockdate и извлекается первый дочерний узел.
* Данные записываются в файл с использованием FileWriter. Устанавливается режим автоматического закрытия файла с помощью оператора try-with-resources.



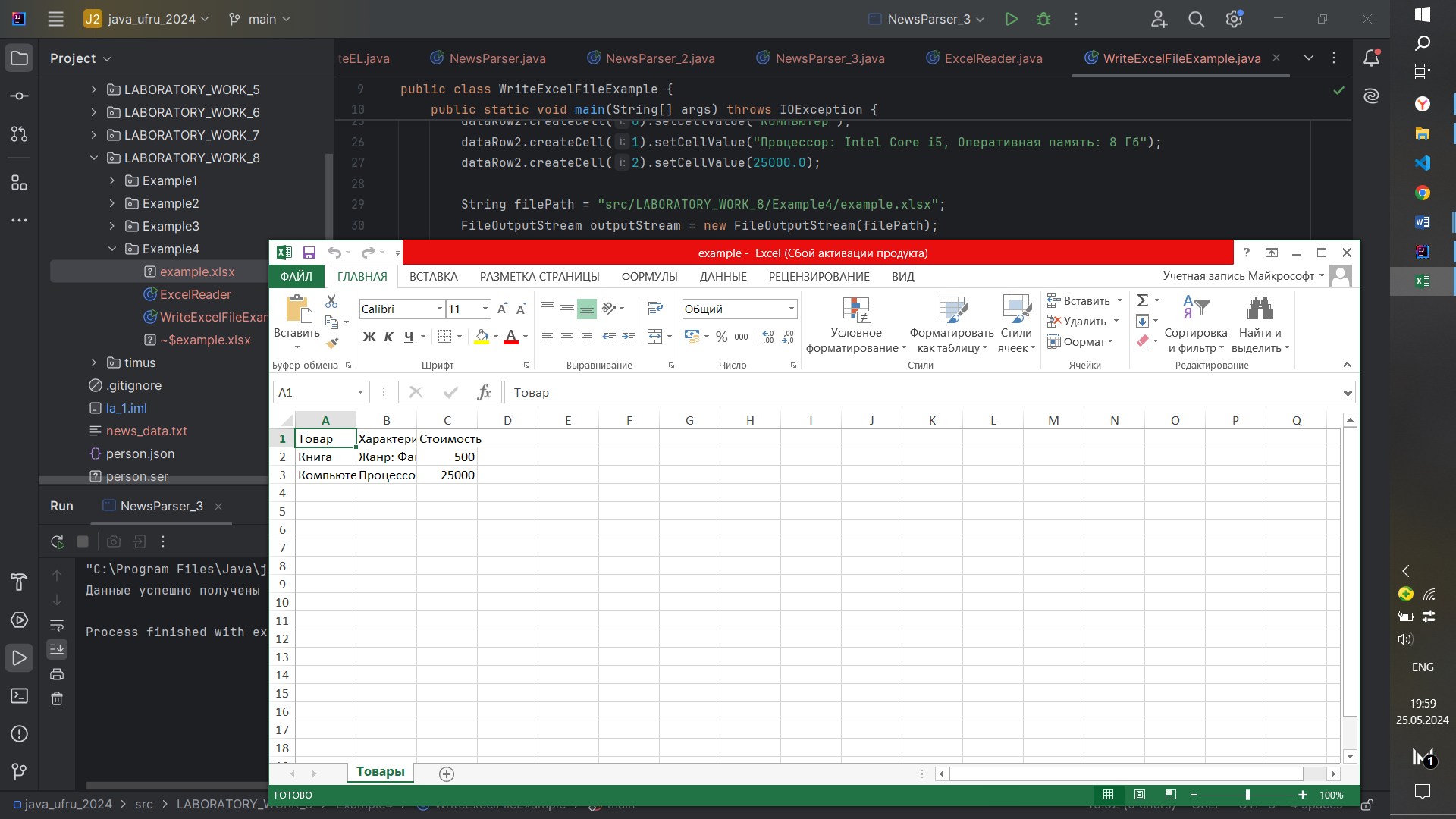
Example4

public class WriteExcelFileExample {  
 public static void main(String[] args) throws IOException {  
 XSSFWorkbook workbook = new XSSFWorkbook();  
 XSSFSheet sheet = workbook.createSheet("Товары");  
  
 Row headerRow = sheet.createRow(0);  
 headerRow.createCell(0).setCellValue("Товар");  
 headerRow.createCell(1).setCellValue("Характеристики");  
 headerRow.createCell(2).setCellValue("Стоимость");  
  
 Row dataRow1 = sheet.createRow(1);  
 dataRow1.createCell(0).setCellValue("Книга");  
 dataRow1.createCell(1).setCellValue("Жанр: Фантастика, Автор: Иванов И.И.");  
 dataRow1.createCell(2).setCellValue(500.0);  
  
 Row dataRow2 = sheet.createRow(2);  
 dataRow2.createCell(0).setCellValue("Компьютер");  
 dataRow2.createCell(1).setCellValue("Процессор: Intel Core i5, Оперативная память: 8 Гб");  
 dataRow2.createCell(2).setCellValue(25000.0);  
  
 String filePath = "src/LABORATORY\_WORK\_8/Example4/example.xlsx";  
 FileOutputStream outputStream = new FileOutputStream(filePath);  
 workbook.write(outputStream);  
 workbook.close();  
 outputStream.close();  
 System.*out*.println("Данные записаны в файл: " + filePath);  
 }  
}

Класс WriteExcelFileExample демонстрирует процесс создания и заполнения файла Excel с использованием библиотеки Apache POI. В этом примере создается новый файл Excel, в который записываются данные о товарах.

* Создание рабочей книги и листа: Создается объект XSSFWorkbook (рабочая книга для формата XLSX) и в ней создается лист с именем "Товары".
* Создание заголовков: Создается первая строка (заголовок) с тремя ячейками, содержащими названия столбцов: "Товар", "Характеристики", "Стоимость".
* Заполнение данными: Создаются две строки данных:
* Первая строка содержит информацию о товаре "Книга" с характеристиками и стоимостью.
* Вторая строка содержит информацию о товаре "Компьютер" с характеристиками и стоимостью.
* Запись файла: Устанавливается путь к файлу Excel и создается FileOutputStream для записи данных в файл. Затем данные из рабочей книги записываются в поток, и закрываются оба потока (workbook и outputStream).

Вывод сообщения: Выводится сообщение о том, что данные успешно записаны в файл.



1. Улучшение работы с ошибками. Программы, которые читают Excel файлы, могут столкнуться с ошибками, такими как неправильный формат файла, отсутствие запрашиваемого листа и т.д. Улучшение работы с ошибками может включать более подробные сообщения об ошибках, рекомендации для исправления ошибок и возможность повторного запуска программы после исправления ошибок.

public class ExcelReader {  
 public static void main(String[] args) {  
 String filePath = "src/LABORATORY\_WORK\_8/Example4/example.xlsx";  
 int maxAttempts = 3;  
 int attempt = 0;  
  
 while (attempt < maxAttempts) {  
 try {  
 FileInputStream fis = new FileInputStream(filePath);  
 Workbook workbook = WorkbookFactory.*create*(fis);  
 Sheet sheet = workbook.getSheetAt(0); // Предполагаем, что нужен первый лист  
 // Обработка листа...  
 fis.close();  
 System.*out*.println("Данные успешно прочитаны.");  
 break;  
 } catch (IOException e) {  
 System.*err*.println("Ошибка при чтении файла: " + e.getMessage());  
 System.*err*.println("Пожалуйста, проверьте путь к файлу и его наличие на диске.");  
 attempt++;  
 } catch (Exception e) {  
 System.*err*.println("Произошла ошибка: " + e.getMessage());  
 attempt++;  
 }  
  
 if (attempt < maxAttempts) {  
 System.*out*.println("Повторная попытка через 5 секунд...");  
 try {  
 Thread.*sleep*(5000); // Пауза между попытками  
 } catch (InterruptedException ex) {  
 Thread.*currentThread*().interrupt();  
 }  
 } else {  
 System.*out*.println("Не удалось прочитать файл после " + maxAttempts + " попыток.");  
 }  
 }  
 }  
}

Класс ExcelReader демонстрирует процесс чтения данных из файла Excel с использованием библиотеки Apache POI. В этом примере код пытается прочитать файл Excel, выполняя повторные попытки в случае ошибок.

* Инициализация пути к файлу и количества попыток: Устанавливается путь к файлу Excel и максимальное количество попыток чтения файла.
* Повторные попытки чтения файла: Используется цикл while, который позволяет повторить попытку чтения файла до 3 раз в случае неудачи.
* Чтение файла Excel:
  + Создается FileInputStream для чтения файла.
  + Используется WorkbookFactory.create(fis) для создания объекта Workbook из потока.
  + Извлекается первый лист (Sheet) из рабочей книги.
  + Обрабатываются данные листа (код обработки данных не приведен в этом примере).
  + Закрывается FileInputStream
* Обработка исключений:
  + Если возникает IOException, выводится сообщение об ошибке, и пытается повторить чтение файла.
  + Если возникает другая ошибка, выводится сообщение об ошибке, и пытается повторить чтение файла.
  + В случае неудачи выполняется повторная попытка с задержкой в 5 секунд.