# Лабораторная работа № 11

**Цель:**

Познакомиться с бинарным хранением данных, углубить знания о стандартных средствах для чтения XML-файлов и манипулирования ими, научиться оформлять проект в виде JAR.

**Постановка задачи:**

**Задание 1.** Разработать XML-формат своих данных. Использовать вложенные теги, хранить данные в текстовых и числовых атрибутах (с перспективой наложения на эти данные ограничений), а также в теле. Реализовать открытие такого xml-файла, разобрав его с помощью DOM, и отобразить в виде таблицы или дерева. Реализовать команду сохранения в XML-формате.

Также должны быть команда удаления выделенного элемента, команда добавления данных. При запуске программы сразу появляется заставка. Оформить в виде JAR. В jar-файле лишнего не хранить.

**Задание 2.** К заданию 1 добавить сохранение и открытие файла в бинарном формате. Для этого использовать интерфейс Serializable и классы ObjectInputStream и ObjectOutputStream.

**Задание 3.** К заданию 1 добавить возможность открытия xml-файла, разобрав его с помощью SAX и выполнить некоторые расчеты (количество, максимум, минимум, среднее или т.п.). Разбор должен осуществлять именно расчеты, а не преобразование в коллекцию. Выполнить не менее двух расчетов.

**Решение задачи:**

**Задание 1.**

Создадим класс, описывающий объекты, информация о которых хранится в XML-файле. Пусть это будет “Planet”. В качестве полей возьмем название планеты, ее массу, диаметр, количество спутников и наличие системы колец.

```
Planet(String name, double mass, double diameter, int count, String system){
    try {
        this.name = name;
        this.mass = mass;
        this.diameter = diameter;
        this.count = count;

        this.system = system.toUpperCase();
    }
    catch (NumberFormatException e){
        throw new MyException(e.getMessage());
    }
}
```

*Рис.1. Конструктор класса “Planet”.*

Файл с информацией оформим следующим образом: корневой узел назовем <planet>, в него поместим атрибут “name”, а для остальных полей класса используем вложенные узлы <mass>, <diameter> и т.д.

```
<planets>
    <planet name="Jupiter">
        <mass>1899</mass>
        <diameter>142984</diameter>
        <count>63</count>
        <ans>YES</ans>
    </planet>
    <planet name="Saturn">
        <mass>568</mass>
        <diameter>120536</diameter>
        <count>47</count>
        <ans>Yes</ans>
    </planet>
    <planet name="Mars">
        <mass>0.642</mass>
        <diameter>6794</diameter>
        <count>2</count>
        <ans>NO</ans>
    </planet>
</planets>
```

*Рис.2. Структура XML-файла.*

Реализуем паттерн проектирования “MVC”. В классе “Controller” определим функции “open” и “write”, используя аналогичный подход: создаем “DocumentBuilderFactory” и “DocumentBuilder”, с помощью которых считываем из файла по очереди атрибуты и узлы каждого объекта, занося их в список “model”.

```java
@Override
public void open(String path) {
    model.clear();
    File xmlFile = new File(path);
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder;
    try {
        builder = factory.newDocumentBuilder();
        Document document = builder.parse(xmlFile);
        document.getDocumentElement().normalize();
        NodeList nodeList = document.getElementsByTagName("planet");

        for (int i = 0; i < nodeList.getLength(); i++) {
            model.add(setPlanet(nodeList.item(i)));
        }

        view.showTable(model);
        view.showInfo( title: "Number of planets",  message: model.size()+"");
    }
    catch (Exception e) {
        view.showError(e.getMessage());
    }
}
```

*Рис.3.Метод для чтения из файла “open”.*

```java
private static Planet setPlanet(Node node) {
    Planet planet;
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element element = (Element) node;
        String name = element.getAttribute( name: "name");
        double mass = Double.parseDouble(getTagValue( tag: "mass", element));
        double diameter = Double.parseDouble(getTagValue( tag: "diameter", element));
        int count = Integer.parseInt(getTagValue( tag: "count", element));
        String ans = getTagValue( tag: "ans", element);
        planet = new Planet(name, mass, diameter, count, ans);

        return planet;
    }
    else {
        return null;
    }
}

private static String getTagValue(String tag, Element element) {
    NodeList nodeList = element.getElementsByTagName(tag).item( index: 0).getChildNodes();
    Node node = nodeList.item( index: 0);
    return node.getNodeValue();
}
```

*Рис.4. Вспомогательные методы для создания объектов из считанных данных.*

По окончании процесса вызывается функция “showTable” класса “View”, в которой создается таблица с полученными данными.

```java
@Override
public void showTable(List<Planet> list){
    planets = FXCollections.observableArrayList(list);
    table=new TableView<>(planets);
    table.setPrefWidth(800);
    nameColumn = new TableColumn<>( text: "Name");
    massColumn = new TableColumn<>( text: "Mass");
    diameterColumn = new TableColumn<>( text: "Diameter");
    countColumn = new TableColumn<>( text: "Satellite count");
    systemColumn = new TableColumn<>( text: "Ring System");
    nameColumn.setCellValueFactory(new PropertyValueFactory<>("name"));
    massColumn.setCellValueFactory(new PropertyValueFactory<>("mass"));
    diameterColumn.setCellValueFactory(new PropertyValueFactory<>("diameter"));
    countColumn.setCellValueFactory(new PropertyValueFactory<>("count"));
    systemColumn.setCellValueFactory(new PropertyValueFactory<>("system"));
    nameColumn.setPrefWidth(250);
    massColumn.setPrefWidth(150);
    diameterColumn.setPrefWidth(150);
    countColumn.setPrefWidth(100);
    systemColumn.setPrefWidth(150);
    table.getColumns().add(nameColumn);
```

*Рис.5. Метод “showTable”.*

Для возможности выбора планеты, которую нужно удалить из списка, создаем “TableViewSelectionModel”. Таким образом, при нажатии на строку таблицы, создается копия выбранной планеты.

```java
private Planet selectedPlanet = null;

private void setSelected(){
    TableView.TableViewSelectionModel<Planet> selectionModel = table.getSelectionModel();
    selectionModel.selectedItemProperty().addListener((val, oldVal, newVal) -> {
        if(newVal != null) {
            selectedPlanet= newVal;
        }
    });
}
```

*Рис.6. Создание модели выбора объекта таблицы.*

Выбранную планету можно убрать из списка,если в дальнейшем кликнуть на кнопку “Remove”. Если нажать на кнопку “Add”, то выскочит новое окно с текстовыми полями ввода ключевых параметров планеты. После корректного ввода (проверка осуществляется с помощью условных операторов и класса “MyException”, генерирующего исключения) будет вызвана функция “update” класса “Controller” и в список добавится новая планета.

```
ok.setOnAction(event1 -> {
    if (nameInput.getText() == null || nameInput.getText().trim().isEmpty()
            || massInput.getText() == null || massInput.getText().trim().isEmpty()
            || diameterInput.getText() == null || diameterInput.getText().trim().isEmpty()
            || countInput.getText() == null || countInput.getText().trim().isEmpty()) {
        showError("Empty field!");
    }
    else {
        try {
            String name = nameInput.getText();
            double mass = Double.parseDouble(massInput.getText());
            double diameter = Double.parseDouble(diameterInput.getText());
            int count = Integer.parseInt(countInput.getText());
            String ans = "No";
            if (yes.isSelected()) {
                ans = "Yes";
            } else if (no.isSelected()) {
                ans = "No";
            }
            if(name.isEmpty() || mass <= 0 || diameter <= 0 || count < 0
                    ||(!ans.toUpperCase().equals("YES") && !ans.toUpperCase().equals("NO")))
                showError("Incorrect input!");
            else{
                Planet addingPlanet = new Planet(name, mass,diameter, count,ans);
                controller.updateList(addingPlanet,  action: "add");
                newStage.close();
            }
```

*Рис.7. Диалоговое окно для добавления новой планеты в таблицу.*

```
@Override
public void updateList(Planet planet, String action){
    switch (action){
        case "add":
            model.add(planet);
            view.showTable(model);
            break;
        case "remove":
            model.remove(planet);
            view.showTable(model);
            break;
    }
}
```

*Рис.8. Метод “updateList”, обновляющий список в зависимости от выбранного действия (удаления.добавления планеты).*

Фон для оконного приложения добавим стандартным образом.

```
Image backgroundImage = new Image(localUrl);
littlePane.setBackground(new javafx.scene.layout.Background(new BackgroundImage(backgroundImage, BackgroundRepeat.NO_REPEAT,
        BackgroundRepeat.NO_REPEAT,
        BackgroundPosition.CENTER,
        BackgroundSize.DEFAULT)));
```

*Рис.9.Стандартный метод задания фонового изображения панели.*

Далее согласно инструкциям создадим jar-файл для запуска приложения без доступа к проекту из среды разработки.

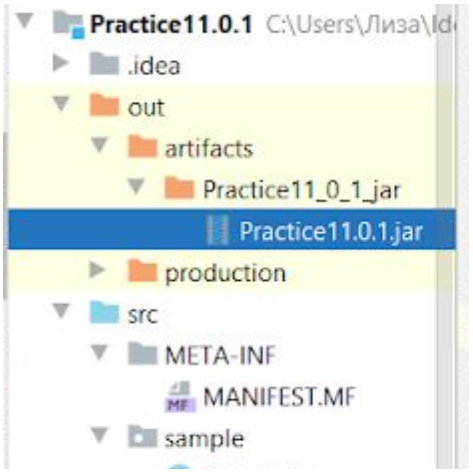


*Рис.10. Созданный JAR.*

**Задание 2.**

Создадим дополнительные вкладки в меню, после нажатия на которые при правильном выборе файла в “FileChooser” вызывается метод контроллера “openDAT” (“writeDAT”).

```
openDAT.setOnAction(event -> {
    FileChooser fileChooser = new FileChooser();
    FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter( description: "DAT files (*.dat)",  ...extensions: "*.dat");
    fileChooser.getExtensionFilters().add(extFilter);
    File f = fileChooser.showOpenDialog(stage);
    if(f!= null){
        controller.openDAT(f.getPath());
    }
    else {
        showError("Incorrect file!");
    }
});
```

*Рис.11. Следствие нажатия на вкладку “openDAT”.*

Для открытия (сохранения) данных в бинарном формате воспользуемся стандартным вводом “ObjectInputStream” (выводом “ObjectOutputStream”).

```java
@Override
public void openDAT(String path){
    try{
        model.clear();
        FileInputStream fileInputStream = new FileInputStream(path);
        ObjectInputStream ois = new ObjectInputStream(fileInputStream);
        model = (ArrayList<Planet>)ois.readObject();
        view.showTable(model);
        view.showInfo( title: "Number of planets",  message: model.size()+"");
    }
    catch (IOException | ClassCastException | ClassNotFoundException ex){
        view.showError(ex.getMessage());
    }
}

@Override
public void writeDAT(String path) {
    try(ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(path))) {
        oos.writeObject(model);
        view.showInfo( title: "You're lucky!",  message: "File saved!");
    }
    catch (Exception ex){
        view.showError(ex.getMessage());
    }
}
```

*Рис.12. Функции чтения/сохранения данных в бинарном формате.*

**Задание 3.**

Считывая информацию с помощью SAX, необходимо сразу же проводить вычисления. Поэтому создадим переменные, хранящие максимальное количество спутников и минимальный диаметр, положим в них минимальное и максимальное значения соответственно. Тогда если у планеты какое-либо из текущих значений меньше/больше, то запоминаем ее название. Таким образом, дойдя до конца документа, получим верные ответы.

```java
@Override
public void SAXCounting(String fileName, String find){
    double minDiameter = 1000000000000.0;
    double diameter = 1;
    int maxCount = 0;
    int count = 0;
    String maxName = "";
    String minName = "";
    String name = "";
    XMLInputFactory xmlInputFactory = XMLInputFactory.newInstance();
```

*Рис.13. Переменные для хранения названий планет и значений их полей.*

```
XMLEvent xmlEvent = reader.nextEvent();
if (xmlEvent.isStartElement()) {
    StartElement startElement = xmlEvent.asStartElement();
    switch (startElement.getName().getLocalPart()) {
        case "planet":
            Attribute namePlanet = startElement.getAttributeByName(new QName( localPart: "name"));
            name = namePlanet.getValue();
            break;
        case "diameter":
            xmlEvent = reader.nextEvent();
            diameter = Double.parseDouble(xmlEvent.asCharacters().getData());

            break;
        case "count":
            xmlEvent = reader.nextEvent();
            count = Integer.parseInt(xmlEvent.asCharacters().getData());
            break;
    }
```

*Рис.14. Обход необходимых узлов объекта.*

```
    if (xmlEvent.isEndElement()) {
        EndElement endElement = xmlEvent.asEndElement();
        if (endElement.getName().getLocalPart().equals("planet")) {
            if(diameter < minDiameter){
                minDiameter = diameter;
                minName = name;
            }
            if(count > maxCount){
                maxCount = count;
                maxName = name;
            }
        }
    }
}
if(find.equals("max")){
    view.showInfo( title: maxName+ " has the most satellites.\n",  message: "Value: " + maxCount);
}
else{
    view.showInfo( title: minName + " has the smallest diameter.\n" ,  message: "Value: " + minDiameter);
}
```

*Рис.15. Проведение вычислений без создания коллекций и вызов соответствующих требованию пользователя функций класса "View" с передачей полученных результатов.*

**Результат:**

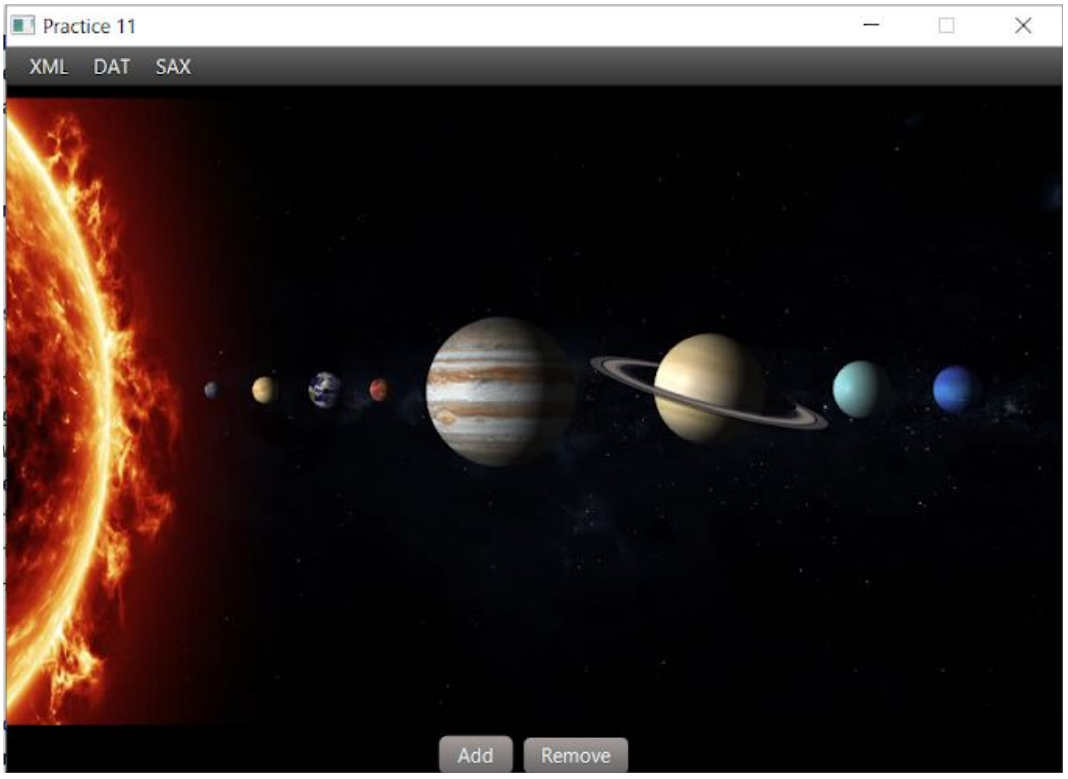


*Рис.16.Вид оконного приложения после запуска.*

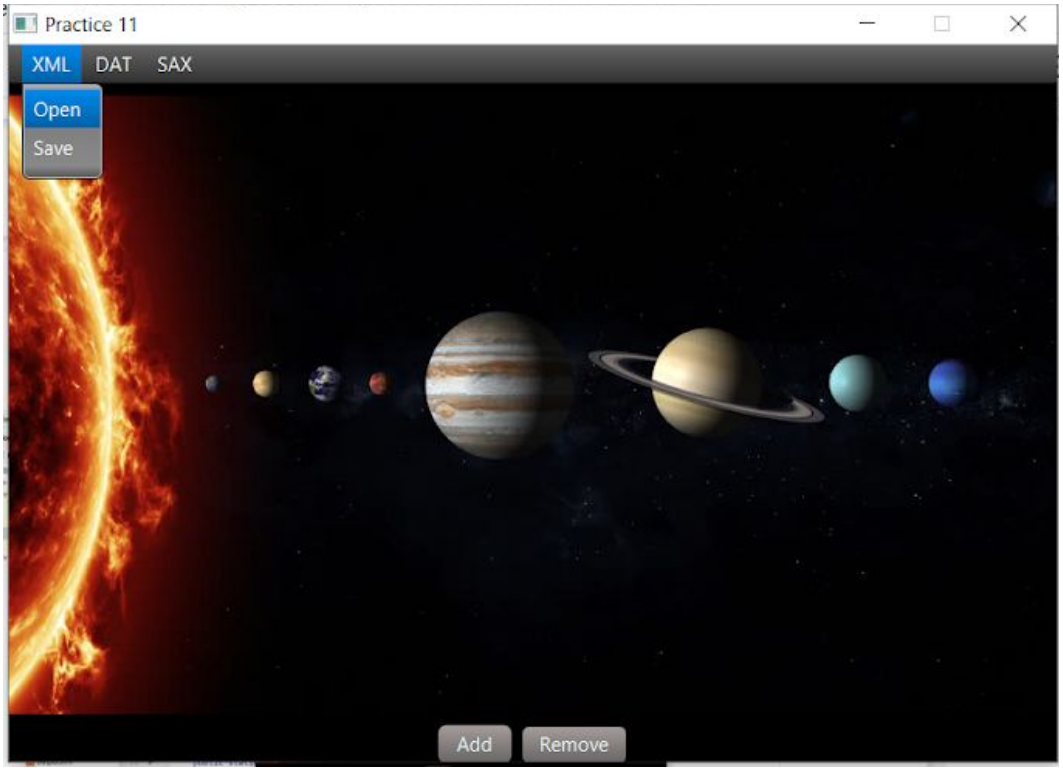


*Рис.17.Демонстрация открытия XML-файла.*

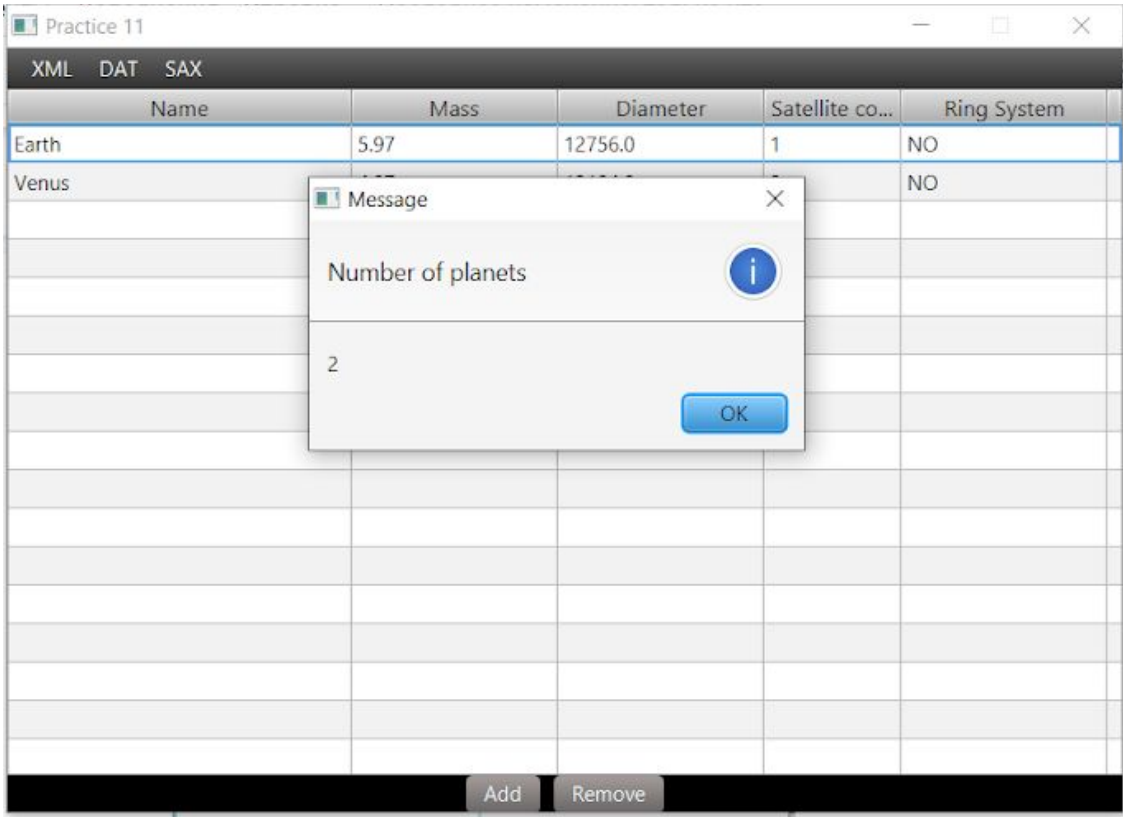


*Рис.18.Демонстрация занесения данных в таблицу.*

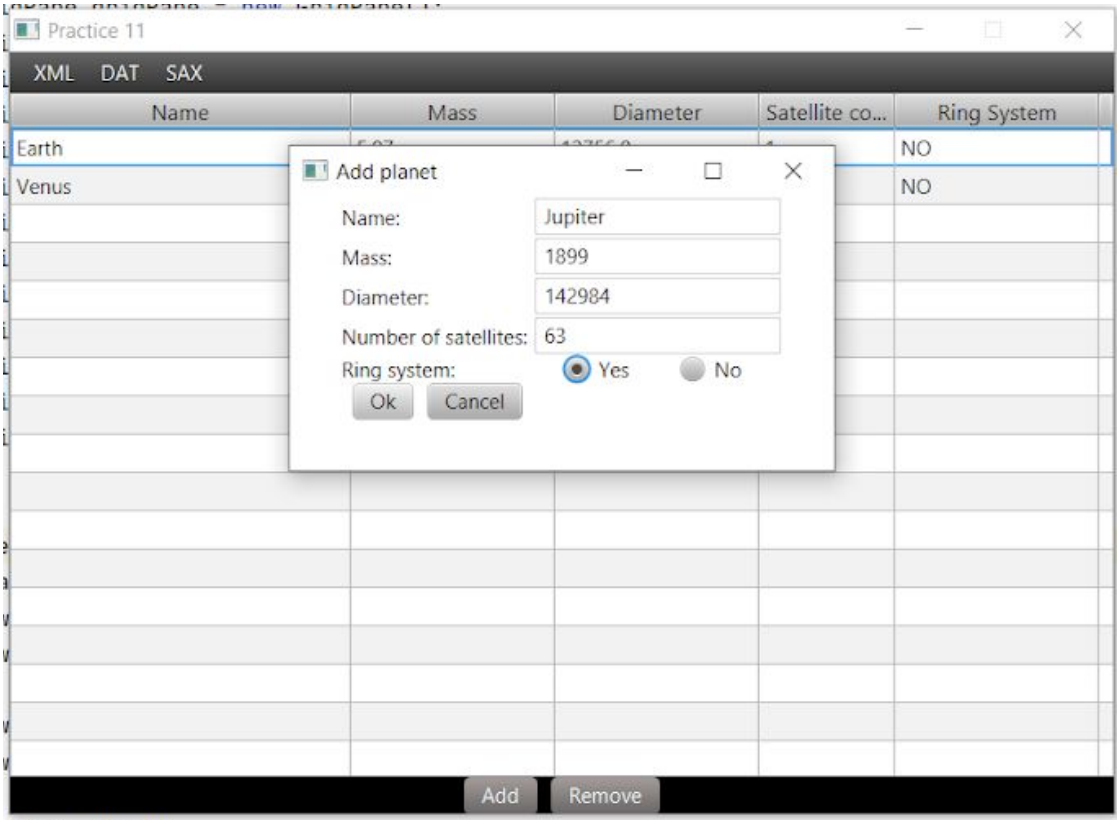


*Рис.19.Демонстрация работы функции добавления элемента в список.*

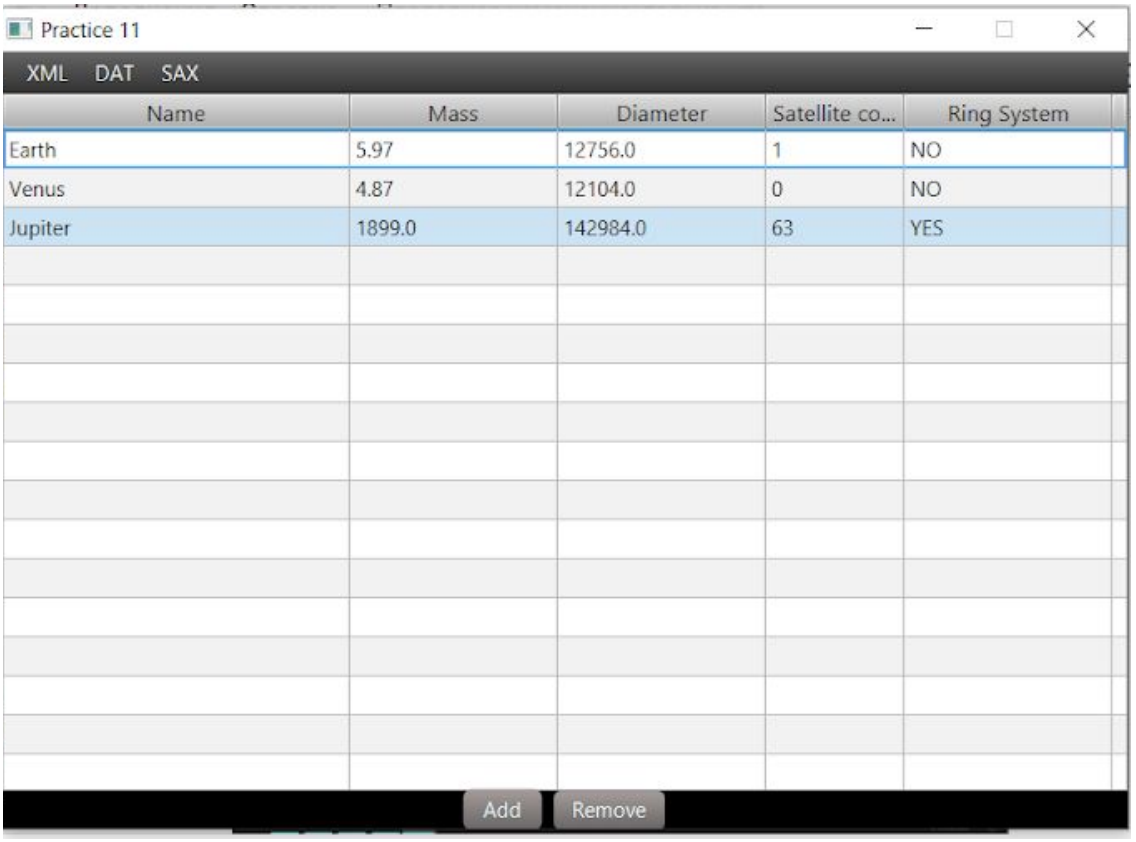


*Рис.20.Занесение добавленной планеты в таблицу.*

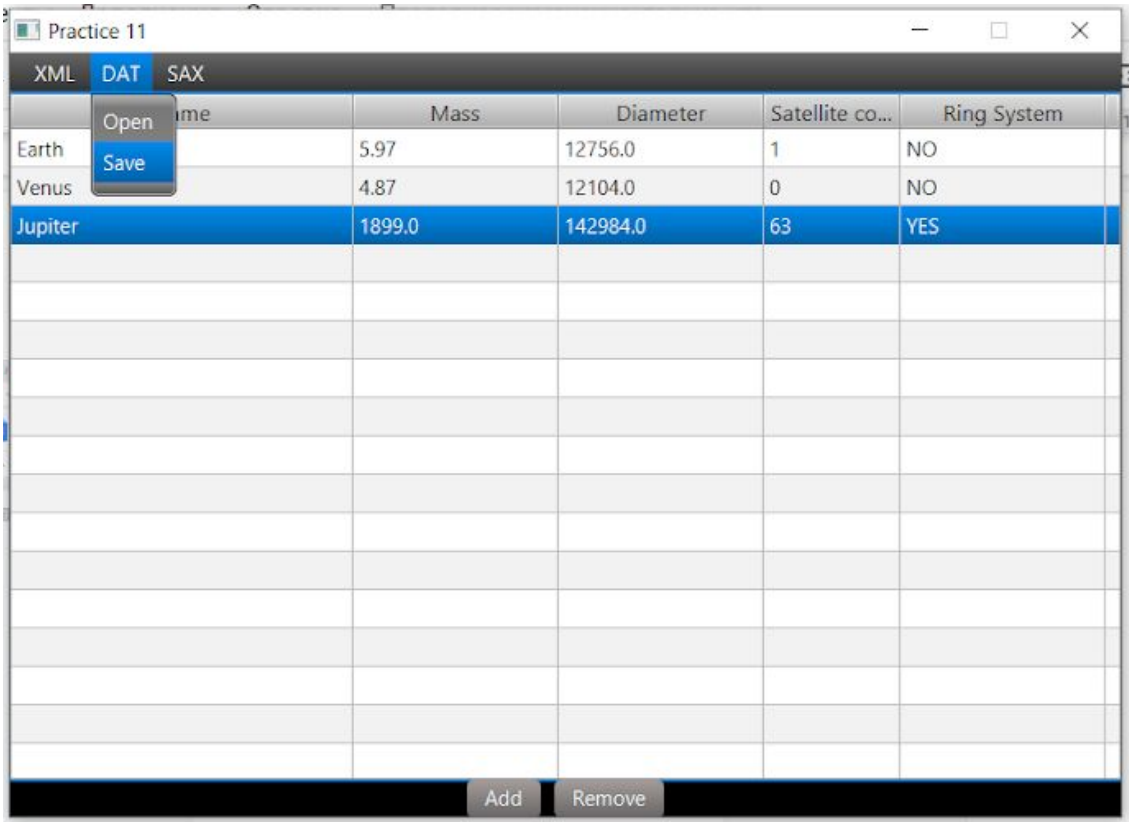


*Рис.21.Демонстрация сохранения данных в бинарном формате.*

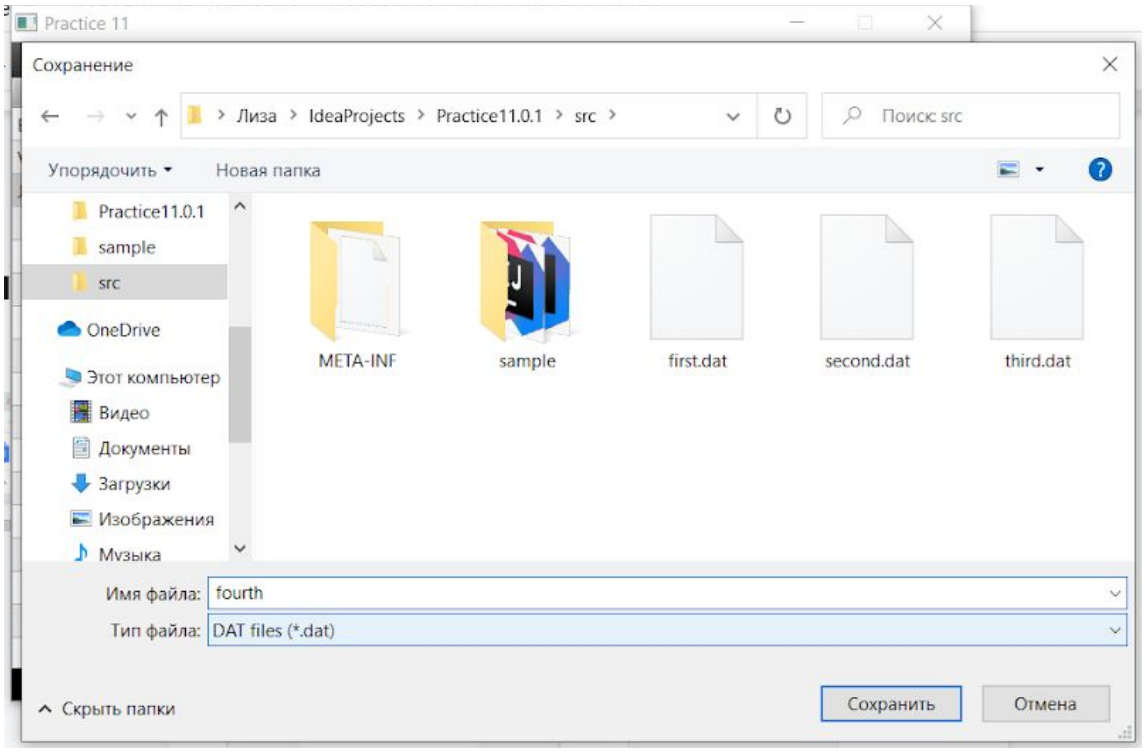


*Рис.22.Сохранения DAT-файла.*

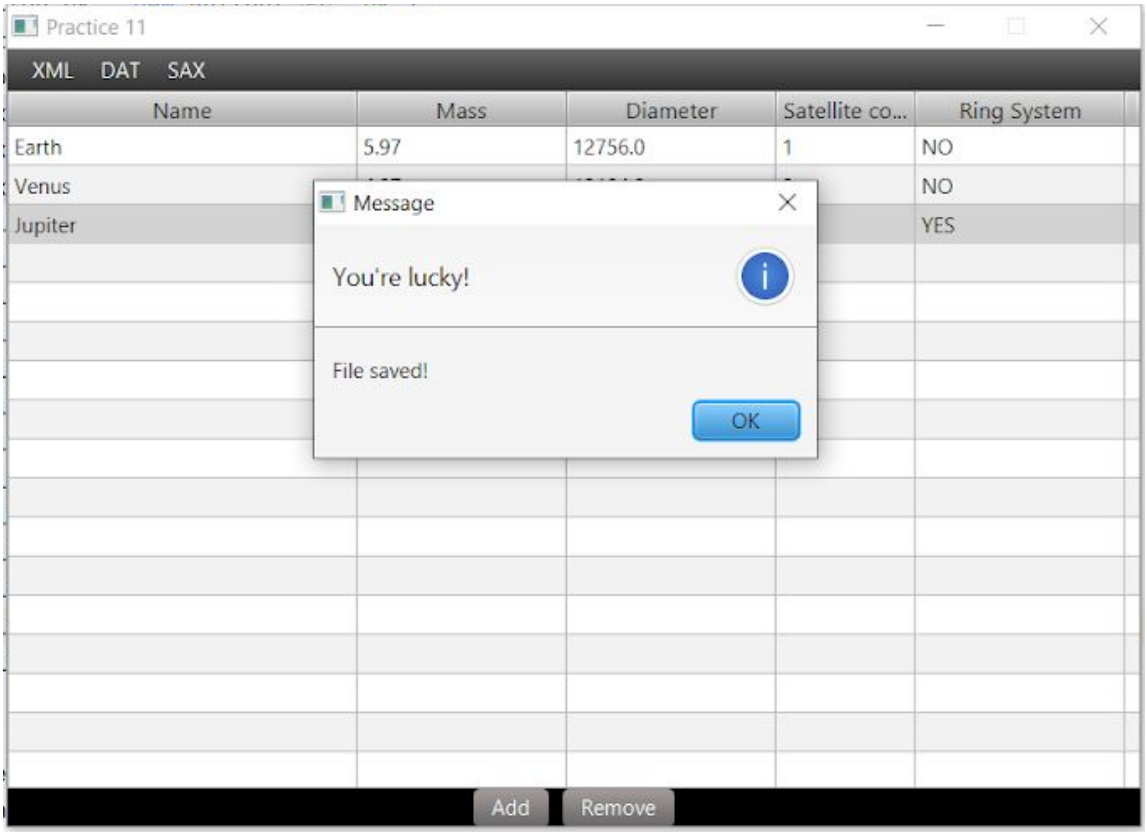


*Рис.23.Оповещение об удачном завершении процедуры сохранения.*

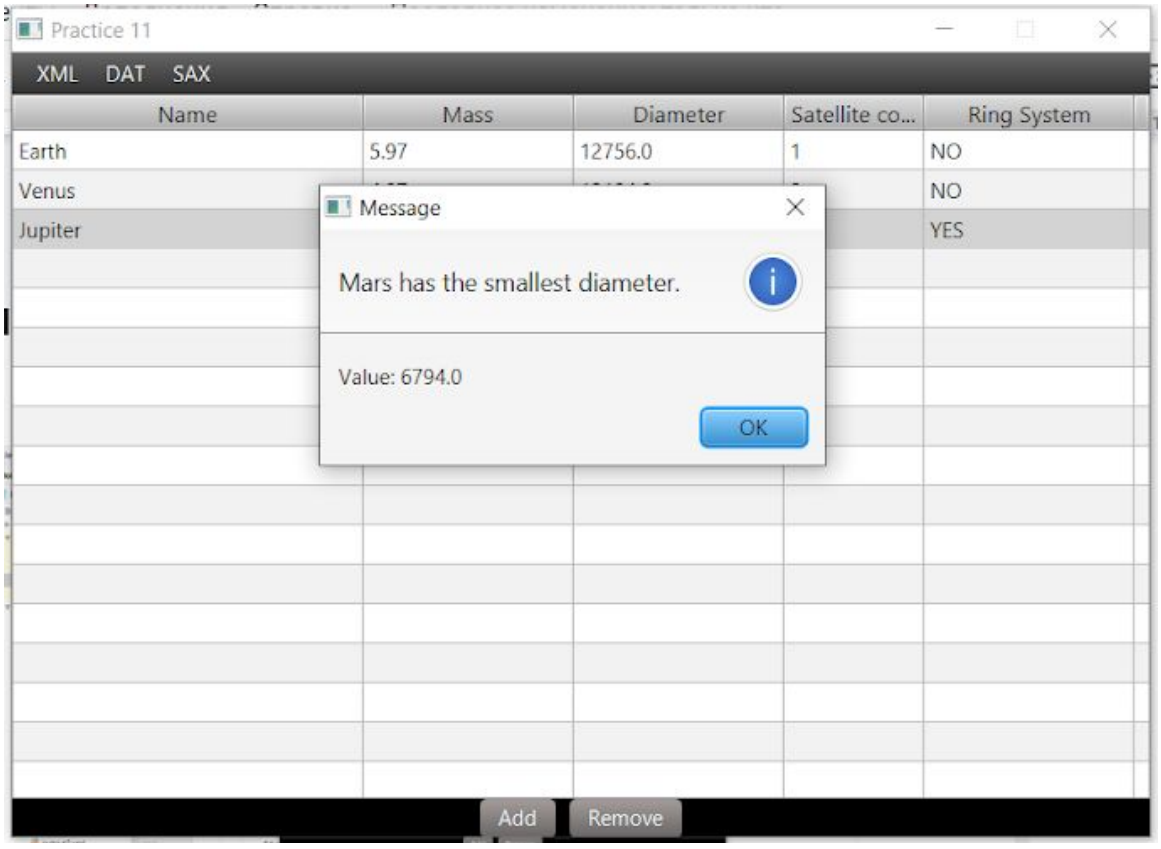


*Рис.24.Демонстрация нахождения планеты с минимальным диаметром.*

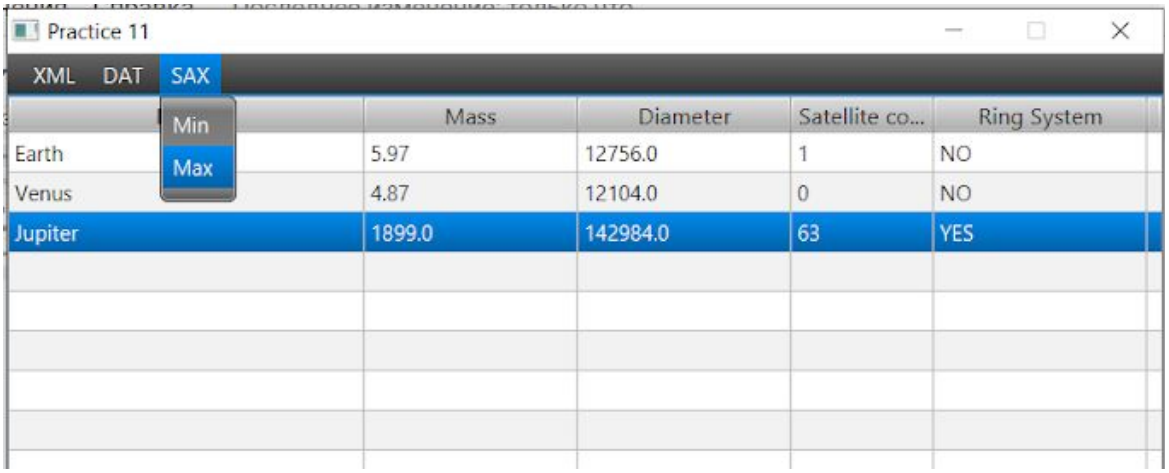


*Рис.25.Демонстрация работы поиска планеты с максимальным количеством спутников.*

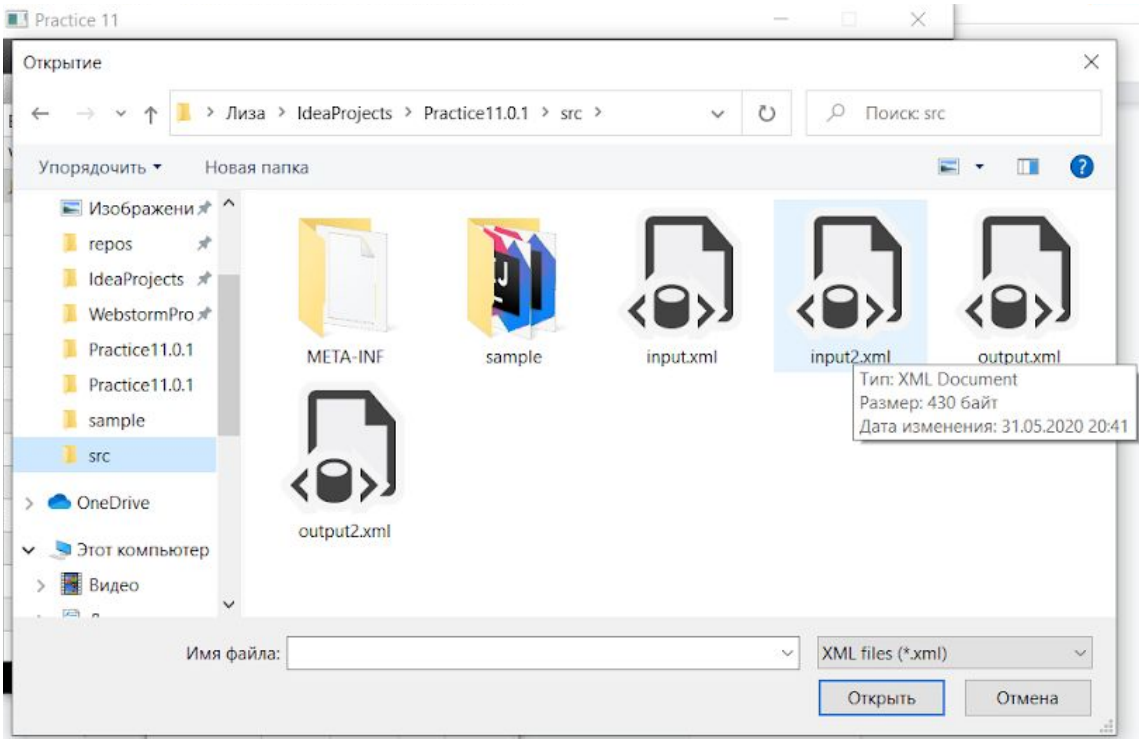


*Рис.26.Открытие XML-файла, в котором нужно провести расчеты.*

.

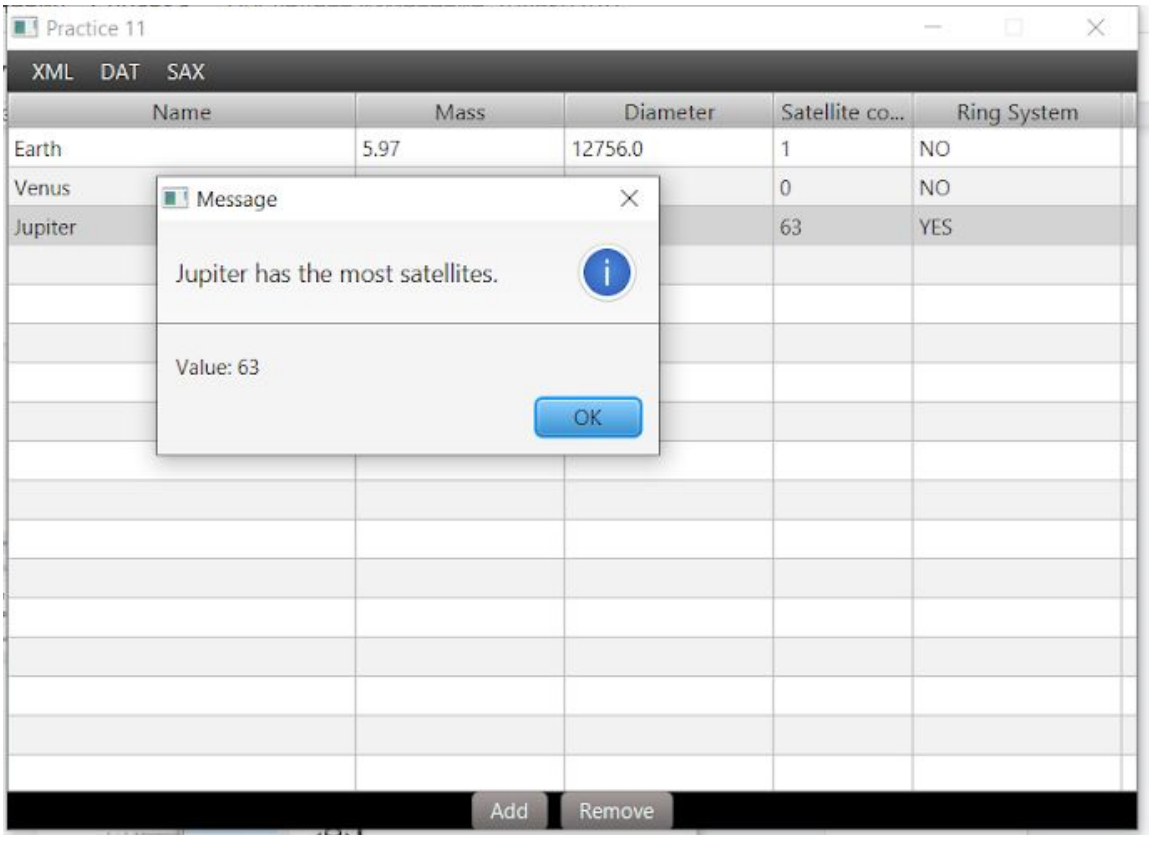


*Рис.27.Демонстрация нахождения планеты с наибольшим количеством спутников.*