**Description of the Code 1**. Personal information:

The provided C# code is a console application that fetches user data from a JSON API endpoint, processes the data, and saves it as an Excel file while also displaying the data in tabular form on the console.

Here is a step-by-step breakdown of the code's functionality:

1. The code starts by defining the necessary using statements, including libraries for working with HTTP, JSON data, and Excel files.
2. The **Main** method is defined as an asynchronous method since it involves asynchronous operations.
3. The API endpoint URL (**https://jsonplaceholder.typicode.com/users**) is stored in a variable named **url**. This endpoint returns user data in JSON format.
4. The code sets up an **HttpClient** to send HTTP requests to the API.
5. The code makes an asynchronous GET request to the API using the **HttpClient.GetAsync** method and awaits the response.
6. If the HTTP response is successful (status code 200-299), the code proceeds to process the retrieved JSON data.
7. The JSON response content is read as a string using **ReadAsStringAsync** method of the **response.Content** property.
8. The JSON data is then parsed into a **JsonDocument** using **JsonDocument.Parse** method to enable easy access to its elements.
9. The code sets up a **StringBuilder** named **builder** to construct the tabular representation of the user data.
10. The tabular header is added to the **builder** containing columns for "Name," "Email," "Phone," and "Address," using **String.Format** to align the columns properly.
11. A line of dashes is added to separate the header from the data, creating a visual separation.
12. An **ExcelPackage** is created to store the user data in an Excel file.
13. A new worksheet named "Users" is added to the **ExcelPackage**.
14. Column headers ("ID," "Name," "Email," "Phone," and "Address") are set in the first row of the worksheet.
15. The JSON data is iterated over using a **foreach** loop to extract individual user properties (ID, Name, Email, Phone, and Address).
16. Each user's data is appended to the **builder** in tabular format, and the user data is written to the corresponding row in the Excel sheet.
17. The Excel cells' width is adjusted automatically to fit the content using **worksheet.Cells.AutoFitColumns()**.
18. The Excel package is saved to a specified file path (**C:\Georgian\Geogrian exept Architecture VM\Encoding\Asgn 3\Users.xlsx**).
19. The tabular representation of user data is printed on the console using **Console.WriteLine(builder.ToString())**.
20. If the API request fails (non-successful status code), an error message is displayed on the console.

Overall, this code demonstrates how to consume a JSON API, process the retrieved data, and create an Excel file with the user data. It also shows how to display the data in tabular form for visual presentation. The code uses the HttpClient for web requests, JSON parsing using JsonDocument, and OfficeOpenXml library for working with Excel files.

**Description of the Code 2**. Books:

This C# code is a console application designed to fetch data from three different sources (JSON, XML, and CSV) over the internet(my GitHub repo. in that particular case), process the data, and then write it to Excel files. The code utilizes several external libraries like **HttpClient**, **Newtonsoft.Json**, **System.Xml.Linq**, **CsvHelper**, and **OfficeOpenXml**.

Here's a step-by-step description of the code:

1. The **Book** class defines a data structure to represent book information, including properties like Category, Title, Author, Year, and Price.
2. The **Program** class contains the **Main** method, which serves as the entry point of the application.
3. Inside the **Main** method, the URLs for the JSON, XML, and CSV data sources are provided.
4. The user is prompted to enter a directory where the output Excel files will be saved.
5. The code uses asynchronous methods (**ProcessJsonAsync**, **ProcessXmlAsync**, and **ProcessCsvAsync**) to fetch data from the respective URLs and create lists of **Book** objects.
6. The **WriteToExcel** method is responsible for writing the book data to Excel files. It uses the **OfficeOpenXml** library to create and manipulate Excel spreadsheets.
7. The code processes the JSON data by parsing it using the **Newtonsoft.Json** library, extracting relevant book information, and displaying it on the console.
8. The code processes the XML data by parsing it using the **System.Xml.Linq** library, extracting book information, and displaying it on the console.
9. The code processes the CSV data using the **CsvHelper** library, extracting book information, and displaying it on the console.
10. The code then writes each set of book data (JSON, XML, CSV) to separate Excel files.
11. Finally, the code combines all the book data from the different sources and writes it to a single Excel file named "books.xlsx" in the specified directory.

Overall, this C# console application demonstrates how to retrieve data from various sources, parse and process it, and then export the data to Excel files for further analysis or reporting. It is a practical example of working with different data formats and handling asynchronous operations to enhance performance and responsiveness.  
  
**Description of the Code 3**. Convert JSON data into XML .  
The provided C# code is a program designed to convert JSON data into XML format. It utilizes the System.IO, System.Text.Json, and System.Xml.Linq namespaces to read JSON data from a file, parse it into a JsonDocument object, and then convert it into an XML representation using XDocument.

Here's how the program works:

1. The program begins in the Main method, where it defines the file paths for the JSON and XML files. The JSON file path is specified as "books.json," and the XML file path is specified as "books.xml."
2. It then reads the content of the JSON file into a string using the File.ReadAllText method.
3. The JSON string is then parsed into a JsonDocument object using the JsonDocument.Parse method. This allows the program to work with the JSON data in a structured manner.
4. Next, the program calls the JsonToXml method and passes the root element of the JsonDocument. The JsonToXml method is responsible for converting the JSON data into an XML format.
5. The JsonToXml method initializes an XDocument object, which will represent the XML data.
6. Inside the ConvertToJson method, the program recursively traverses the JSON data. It checks the type of each JSON element (object, array, or scalar value) and handles it accordingly.
7. For JSON objects, the program iterates through the properties of the object and recursively calls the ConvertToJson method to convert each property.
8. For JSON arrays, the program iterates through the elements of the array and recursively calls the ConvertToJson method to convert each array element.
9. For scalar values (string, number, boolean), the program sets the value of the corresponding XML element directly.
10. After processing all JSON elements, the ConvertToJson method adds the constructed XML element to its parent in the XDocument.
11. Finally, the program saves the generated XML data into the specified XML file using the xDocument.Save method.

In summary, this C# program provides a straightforward approach to convert JSON data to XML using recursive methods and demonstrates how to work with JSON and XML data structures in C#.

**Description of the Code 4**. Convert XML data into JSON.

The provided C# program demonstrates a bidirectional conversion between XML and JSON formats. The main goal of this program is to read an XML file, convert its content to a JSON representation, and save it into a separate JSON file. Additionally, it allows converting JSON data back to XML and printing the JSON output.

The program consists of the following components:

1. Main() method: This is the entry point of the program. It defines file paths for the JSON and XML files. It reads the content of the JSON file, parses it using JsonDocument, and converts it to an XDocument (XML representation). Then, it saves the generated XML to the specified file. Finally, it converts the XML back to JSON using XmlToJson() method and prints the resulting JSON.
2. JsonToXml() method: This method takes a JsonElement object as input, recursively traverses through its properties, arrays, and scalar values, and constructs an XDocument (XML representation). It returns the generated XDocument.
3. ConvertToJson() method: This is a recursive method that converts JSON elements to XML elements. It takes a JsonElement, an XContainer (parent element in the XML document), and a string name (element name) as inputs. It checks the value kind of the JSON element and processes it accordingly. For JSON objects, it iterates through properties and converts each one. For JSON arrays, it iterates through elements and converts each one. For scalar values (string, number, boolean), it sets the element's value directly. The constructed element is then added to its parent in the XML document.
4. XmlToJson() method: This method takes an XDocument as input, recursively traverses through its elements, and constructs a JSON representation using a custom JsonObject class. It converts the XML elements to a dictionary representation, and then uses JsonSerializer to serialize the dictionary to a JSON string.
5. JsonObject class: This custom class represents a JSON object as a dictionary with key-value pairs. It overloads the index operator ([]) to allow accessing and setting key-value pairs in the dictionary. It also overrides the ToString() method to convert the dictionary to a properly indented JSON string.

Overall, this C# program provides a useful utility to convert XML data to JSON format and vice versa, showcasing the bidirectional conversion capabilities and demonstrating how to handle complex JSON and XML structures.