Keretrendszer alapú programozás

Fazekas Csaba

fazekas.csaba@uni-eszterhazy.hu

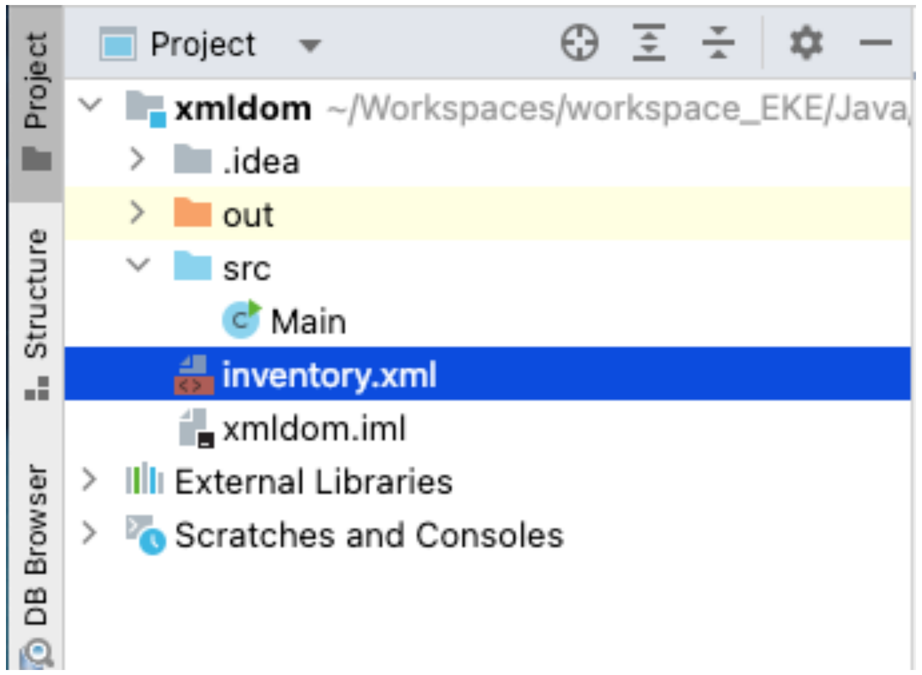
XML Parser

XML Elemző (Parser)

- XML dokumentumok feldolgozására szolgálnak.
- Széles körben a következőket szokták használni:

- **Dom Parser** – Teljes XML dokumentumot betölti, majd elkészíti a teljes hierarchikus reprezentációját a memóriában.
- **SAX Parser** – Események hatására a dokumentum szükséges részleteit töltik csak be.
- **JDOM Parser** – Egyszerűbb DOM feldolgozó.
- **StAX Parser** – Egyszerűbb SAX feldolgozó.
- **XPath Parser** – Parses an XML document based on expression and is used extensively in conjunction with XSLT.
- **DOM4J Parser** – A java library to parse XML, XPath, and XSLT using Java Collections Framework. It provides support for DOM, SAX, and JAXP.

XML Példa



xmldom ~/Workspaces/workspace_EKE/Java
- .idea
- out
- src
  - Main
- inventory.xml
- xmldom.iml
- External Libraries
- Scratches and Consoles

```xml
<?xml version = "1.0"?>
<inventory>
    <product id = "00000001">
        <name>Cola</name>
        <type>Beverage</type>
        <quantity>10</quantity>
    </product>

    <product id = "00000002">
        <name>Tuna Sandwich</name>
        <type>Food</type>
        <quantity>4</quantity>
    </product>

    <product id = "00000003">
        <name>Salad</name>
        <type>Fresh Food</type>
        <quantity>0</quantity>
    </product>
</inventory>
```

Java XML feldolgozó

```
public class Main {
    public static void main(String[] args) {

        // DocumentBuilder : "Defines a factory API that enables
        // applications to obtain
        // a parser that produces DOM object trees from XML documents."
        DocumentBuilderFactory factory =
                DocumentBuilderFactory.newInstance();

        try {
            // The file we will load
            File inputFile = new File("inventory.xml");
            // We should create a documentum that represents the XML file
            DocumentBuilderFactory documentBuilderFactory =
                        DocumentBuilderFactory.newInstance();
            DocumentBuilder documentBuilder =
                        documentBuilderFactory.newDocumentBuilder();
            Document doc = documentBuilder.parse(inputFile);
            doc.getDocumentElement().normalize();
            // Add processing here . . .
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Java XML feldolgozó

```
// Get the root element
String rootElement = doc.getDocumentElement().getNodeName();
System.out.println("The root element is: " + rootElement);

// Information about the XML file
System.out.println("XML version: "+doc.getXmlVersion());
System.out.println("XML encoding: "+doc.getXmlEncoding());
System.out.println("URI: "+doc.getDocumentURI());

// NodeList is an ordered collection of nodes - collect all product
elements
NodeList nodeList = doc.getElementsByTagName("product");
```

Java XML feldolgozó

```java
for (int index = 0; index < nodeList.getLength(); index++
    Node node = nodeList.item(index);

    // We can retrieve information about the node
    System.out.println("\nElement name: " +
        node.getNodeName());

    // We are interested in elements
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element element = (Element) node;

        System.out.println("  Product id: "
                + element.getAttribute("id"));

        System.out.println("  Product name: "
                + element
                .getElementsByTagName("name")
                .item(0)
                .getTextContent());

        System.out.println("  Product type: "
                + element
                .getElementsByTagName("type")
                .item(0)
                .getTextContent());

        System.out.println("  Quantity: "
                + element
                .getElementsByTagName("quantity")
                .item(0)
                .getTextContent());
    }
}
```

JSON

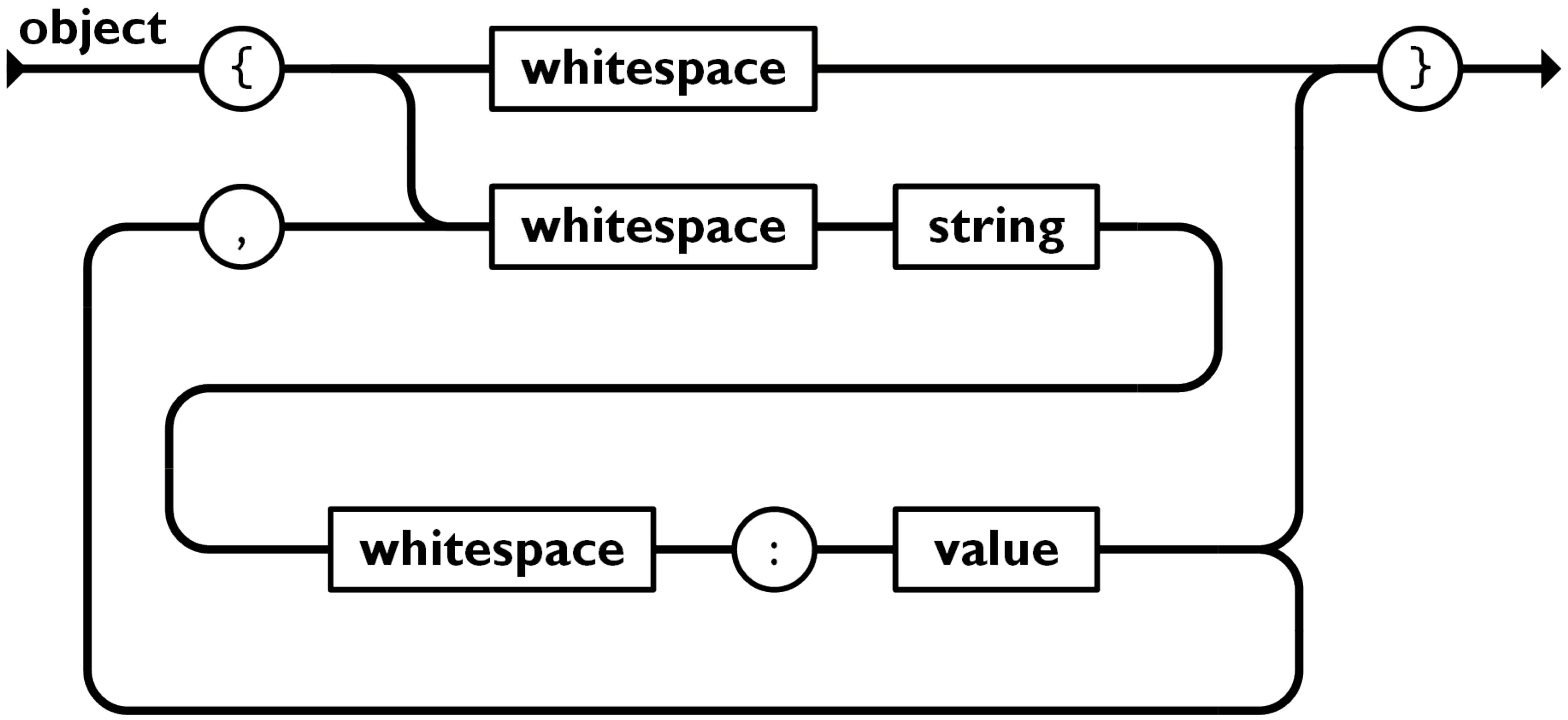
JSON

- Java Script Object Notation - JavaScript objektumjelölés
- Douglas Crockford hozta létre 2000 környékén.
- 2002 - elindul a json.org oldal.
- Egyszerű adatcserét tesz lehetővé (pl XML-hez képest).
- Nyelv független, de felhasznál konvenciókat amik elfogadottak programnyelvekben.

JSON

- Két szerkezetre épül:
  - objektum (kulcs-érték párokként)
  - objektum lista (tömb).

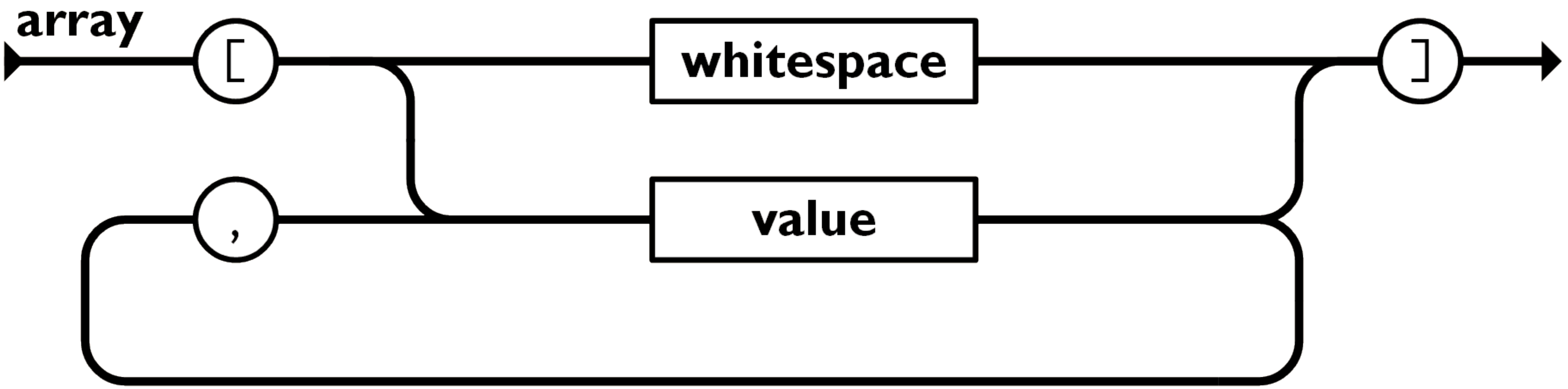
Objektum felépítése



An *object* is an unordered set of name/value pairs. An object begins with **{***left brace* and ends with **}***right brace*. Each name is followed by **:***colon* and the name/value pairs are separated by **,***comma*.

json.org

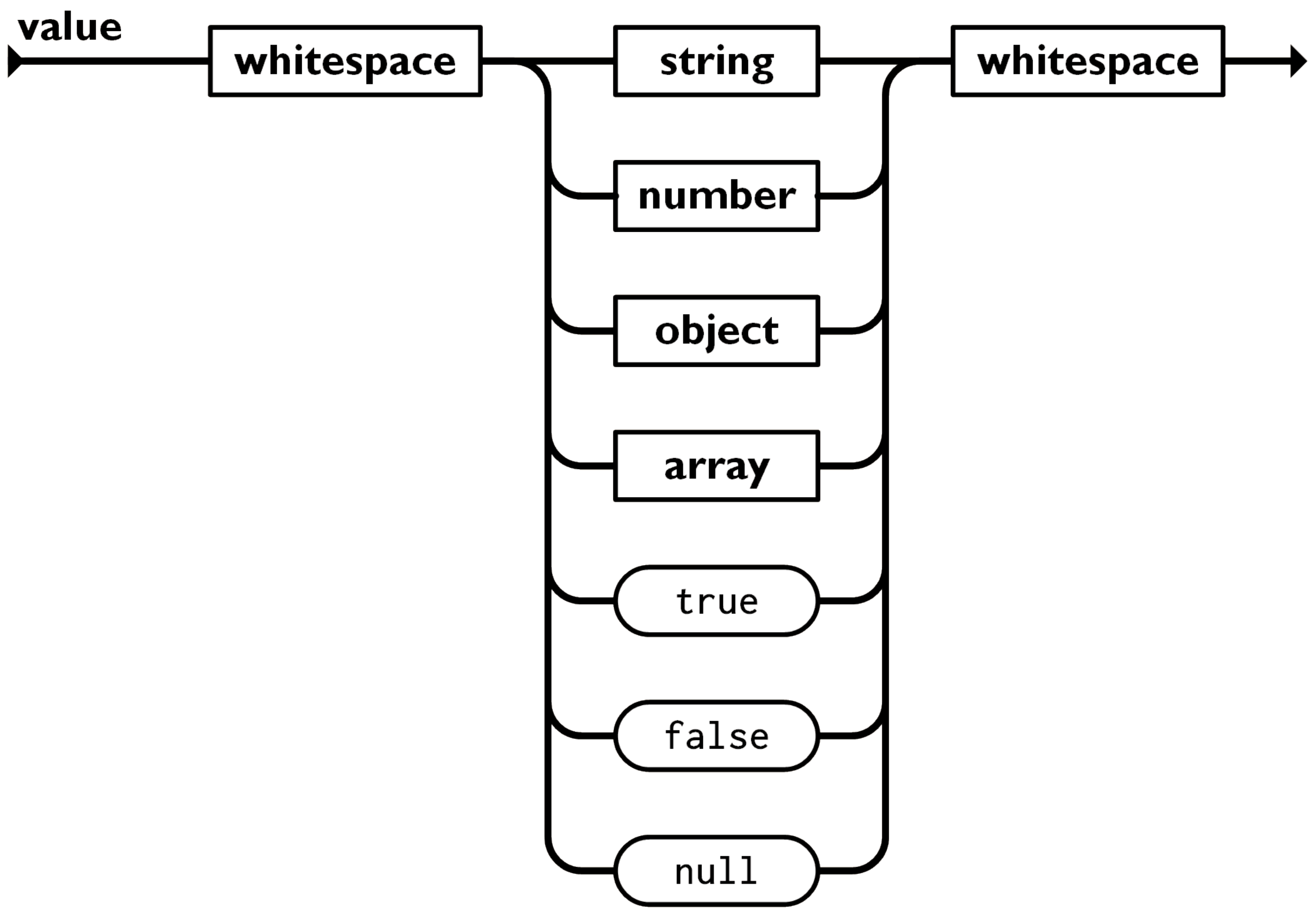
Tömb



An *array* is an ordered collection of values. An array begins with **[***left bracket* and ends with **]***right bracket*. Values are separated by **,***comma*.

json.org

Értékek



A *value* can be a *string* in double quotes, or a *number*, or **true** or **false** or **null**, or an *object* or an *array*. These structures can be nested.

JSON Schema

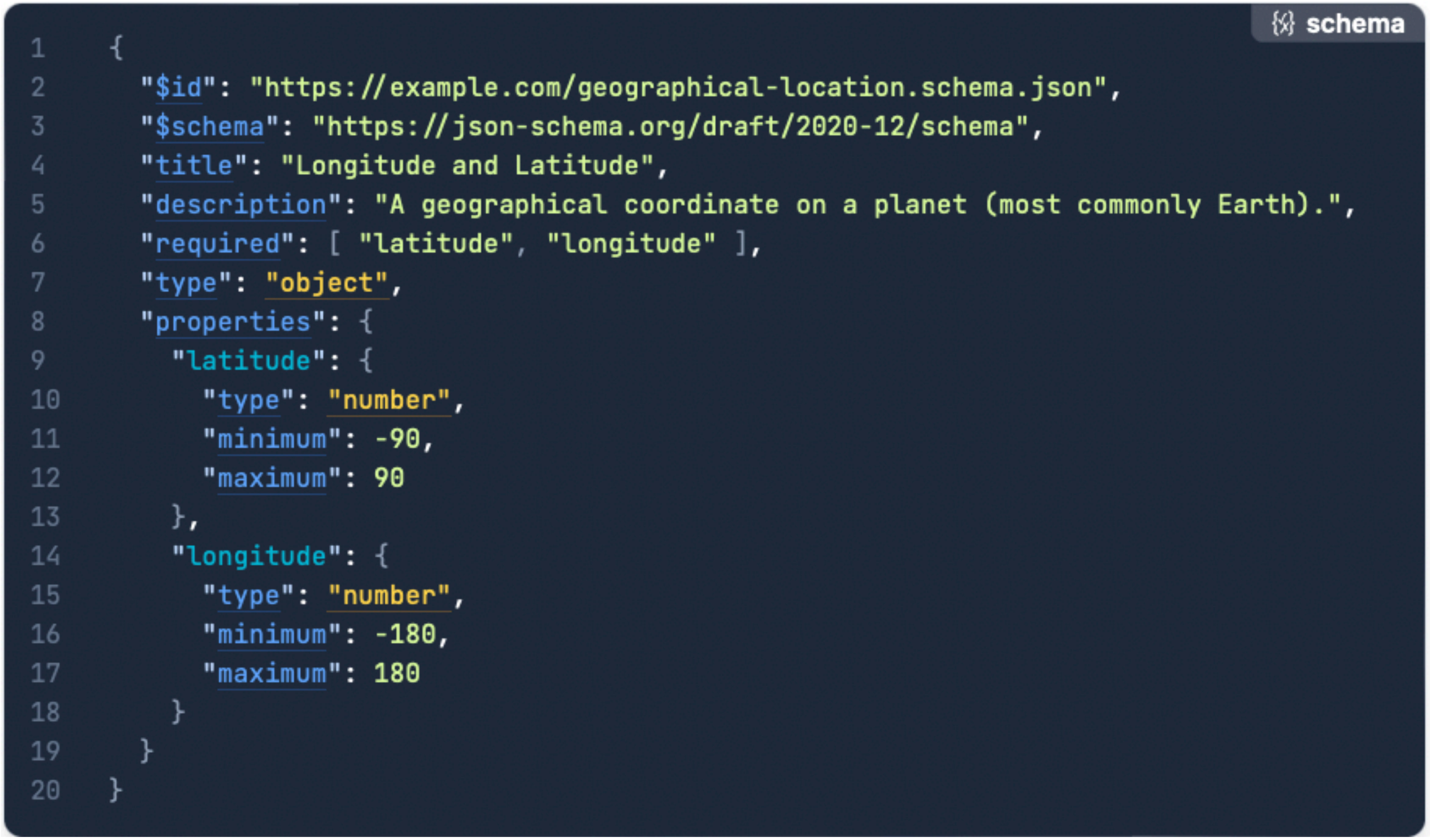
JSON Schema

JSON is probably the most popular format for exchanging data,

JSON Schema is the vocabulary that enables JSON data consistency, validity, and interoperability at scale.

https://json-schema.org/

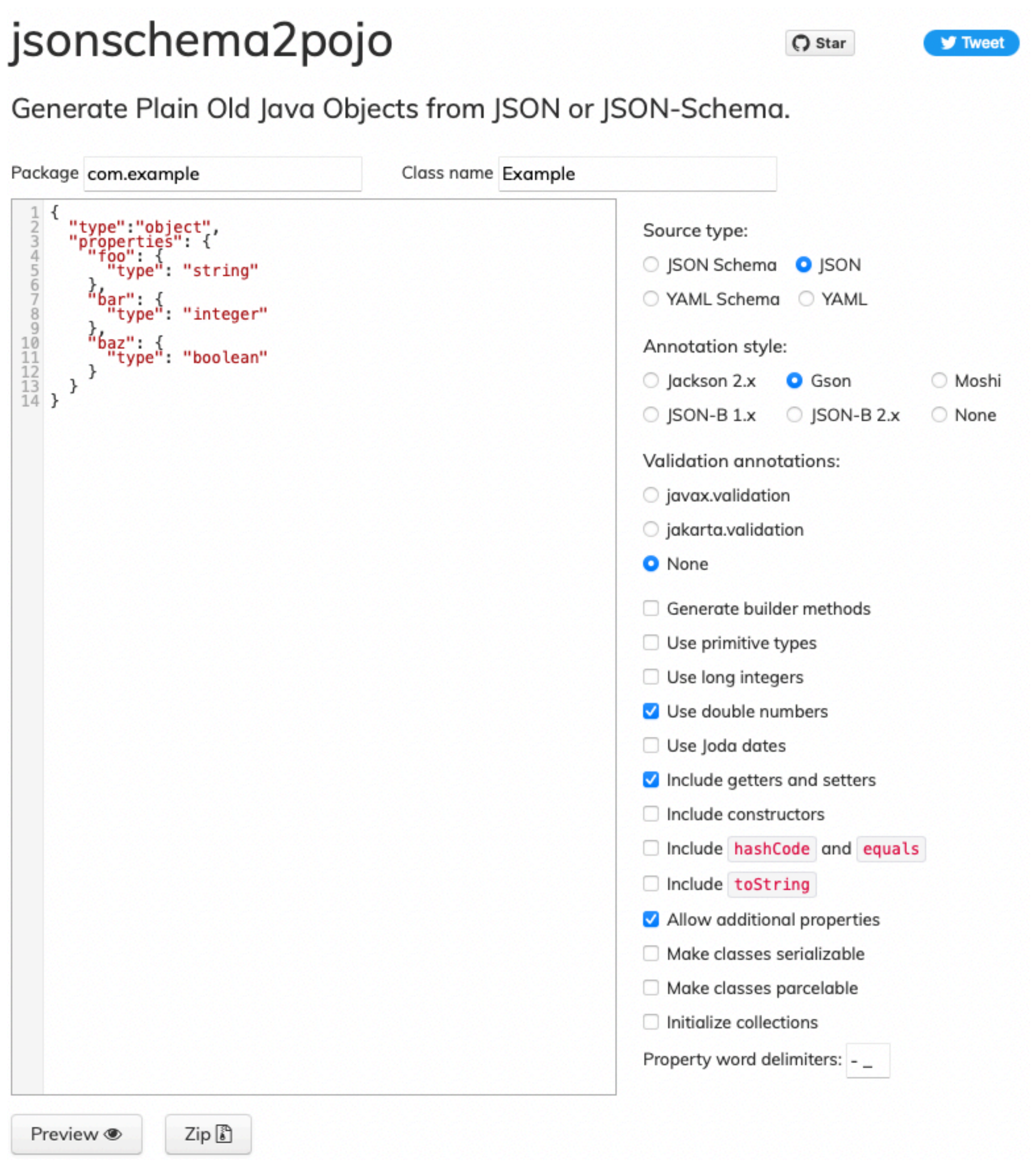
JSON Schema



```json
{
  "$id": "https://example.com/geographical-location.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Longitude and Latitude",
  "description": "A geographical coordinate on a planet (most commonly Earth).",
  "required": [ "latitude", "longitude" ],
  "type": "object",
  "properties": {
    "latitude": {
      "type": "number",
      "minimum": -90,
      "maximum": 90
    },
    "longitude": {
      "type": "number",
      "minimum": -180,
      "maximum": 180
    }
  }
}
```

https://json-schema.org/

Generate Plain Old Java Objects from JSON or JSON-Schema.



Generate Plain Old Java Objects from JSON or JSON-Schema.

Package: com.example   Class name: Example

```
{
  "type":"object",
  "properties": {
    "foo": {
      "type": "string"
    },
    "bar": {
      "type": "integer"
    },
    "baz": {
      "type": "boolean"
    }
  }
}
```

Source type:
JSON Schema [✓ JSON]
YAML Schema YAML

Annotation style:
Jackson 2.x [✓ Gson] Moshi
JSON-B 1.x JSON-B 2.x None

Validation annotations:
javax.validation
jakarta.validation
[✓ None]

- [ ] Generate builder methods
- [ ] Use primitive types
- [ ] Use long integers
- [x] Use double numbers
- [ ] Use Joda dates
- [x] Include getters and setters
- [ ] Include constructors
- [ ] Include `hashCode` and `equals`
- [ ] Include `toString`
- [x] Allow additional properties
- [ ] Make classes serializable
- [ ] Make classes parcelable
- [ ] Initialize collections

Property word delimiters: - _

Preview   Zip

```
{
  "products" : [
    {
      "quantity" : 10,
      "id" : "00000001",
      "name" : "Cola",
      "type" : "Beverage"
    },
    {
      "quantity" : 4,
      "id" : "00000002",
      "name" : "Tuna Sandwich",
      "type" : "Food"
    },
    {
      "quantity" : 0,
      "id" : "00000003",
      "name" : "Salad",
      "type" : "Fresh Food"
    }
  ]
}
```

https://www.jsonschema2pojo.org/

Generate Plain Old Java Objects from JSON or JSON-Schema.

```java
package com.example;

import java.util.List;
import javax.annotation.Generated;

public class Example {

    private List<Product> products;

    public List<Product> getProducts() {
        return products;
    }

    public void setProducts(List<Product> products) {
        this.products = products;
    }
}
```

```json
{
  "products" : [
    {
      "quantity" : 10,
      "id" : "00000001",
      "name" : "Cola",
      "type" : "Beverage"
    },
    {
      "quantity" : 4,
      "id" : "00000002",
      "name" : "Tuna Sandwich",
      "type" : "Food"
    },
    {
      "quantity" : 0,
      "id" : "00000003",
      "name" : "Salad",
      "type" : "Fresh Food"
    }
  ]
}
```

https://www.jsonschema2pojo.org/

Generate Plain Old Java Objects from JSON or JSON-Schema.

```java
package com.example;

import javax.annotation.Generated;

public class Product {

    private Integer quantity;
    private String id;
    private String name;
    private String type;
}
```

```json
{
  "products" : [
    {
      "quantity" : 10,
      "id" : "00000001",
      "name" : "Cola",
      "type" : "Beverage"
    },
    {
      "quantity" : 4,
      "id" : "00000002",
      "name" : "Tuna Sandwich",
      "type" : "Food"
    },
    {
      "quantity" : 0,
      "id" : "00000003",
      "name" : "Salad",
      "type" : "Fresh Food"
    }
  ]
}
```

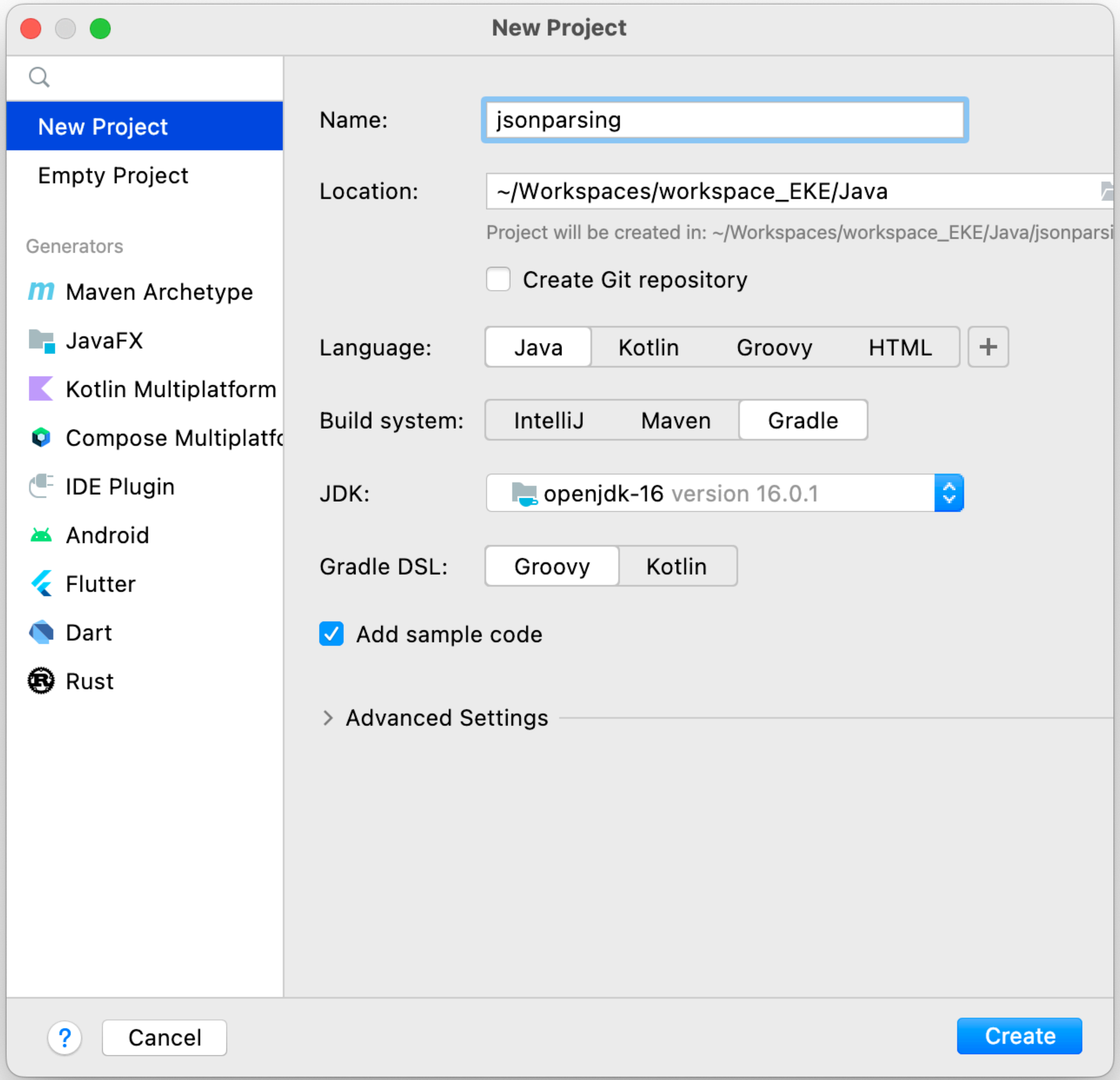
https://www.jsonschema2pojo.org/

Pretty vs Compact printing

```
{
  "products" : [
    {
      "quantity" : 10,
      "id" : "00000001",
      "name" : "Cola",
      "type" : "Beverage"
    },
    {
      "quantity" : 4,
      "id" : "00000002",
      "name" : "Tuna Sandwich",
      "type" : "Food"
    },
    {
      "quantity" : 0,
      "id" : "00000003",
      "name" : "Salad",
      "type" : "Fresh Food"
    }
  ]
}
```

```
{"products":[{"quantity":10,"id":"00000001","name":"Cola","type":"Beverage"},
{"quantity":4,"id":"00000002","name":"Tuna Sandwich","type":"Food"},
{"quantity":0,"id":"00000003","name":"Salad","type":"Fresh Food"}]}
```

Készítsünk egy JSON feldolgozót!


New Project

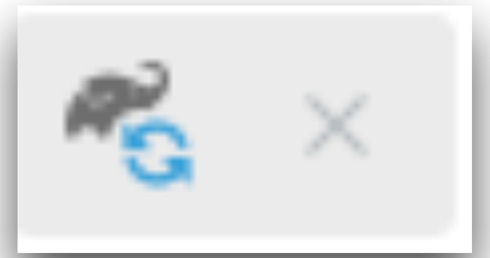
New Project
Empty Project
Generators
Maven Archetype
JavaFX
Kotlin Multiplatform
Compose Multiplatfo
IDE Plugin
Android
Flutter
Dart
Rust

Name: jsonparsing
Location: ~/Workspaces/workspace_EKE/Java
Project will be created in: ~/Workspaces/workspace_EKE/Java/jsonparsi
Create Git repository [ ]
Language: [✓ Java] Kotlin Groovy HTML +
Build system: IntelliJ Maven [✓ Gradle]
JDK: openjdk-16 version 16.0.1
Gradle DSL: [✓ Groovy] Kotlin
Add sample code [✓]
Advanced Settings

? Cancel Create

Készítsünk egy JSON feldolgozót!

```
plugins {
    id 'java'
}

group 'org.example'
version '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.json:json:20230618'

    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'
}

test {
    useJUnitPlatform()
}
```



Készítsünk egy JSON feldolgozót!

```java
import org.json.JSONArray;
import org.json.JSONObject;

public class Main {
    public static void main(String[] args) {

        String jsonString = "{\"products\":[
        {\"quantity\":10,\"id\":\"00000001\",\″name\":\"Cola\",\"type\":\"Beverage\"},
        {\"quantity\":4,\"id\":\"00000002\",\"name\":\"Tuna Sandwich\",\"type\":\"Food\"},
        {\"quantity\":0,\"id\":\"00000003\",\"name\":\"Salad\",\"type\":\"Fresh Food\"}]}";

        JSONObject object = new JSONObject(jsonString);
        JSONArray array = object.getJSONArray("products");

        for (int i = 0; i < array.length(); i++)
        {
            String id = array.getJSONObject(i).getString("id");
            System.out.println(id);

            String name = array.getJSONObject(i).getString("name");
            System.out.println(name);

            String type = array.getJSONObject(i).getString("type");
            System.out.println(type);

            int quantity = array.getJSONObject(i).getInt("quantity");
            System.out.println(String.valueOf( quantity ));
        }
    }
}
```

GSON

GSON

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

There are a few open-source projects that can convert Java objects to JSON. However, most of them require that you place Java annotations in your classes; something that you can not do if you do not have access to the source-code. Most also do not fully support the use of Java Generics. Gson considers both of these as very important design goals.

https://github.com/google/gson

GSON

Gradle:

```
dependencies {
  implementation 'com.google.code.gson:gson:2.10.1'
}
```

Maven:

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10.1</version>
</dependency>
```

https://github.com/google/gson

GSON

```java
import java.util.List;

public class Products {

    public List<Product> products;
}
```

```java
public class Product {

    public Integer quantity;
    public String id;
    public String name;
    public String type;
}
```

GSON

```
String jsonString = "{\"products\":[
        {\"quantity\":10,\"id\":\"00000001\",\”name\":\"Cola\",\"type\":\"Beverage\"},
        {\"quantity\":4,\"id\":\"00000002\",\"name\":\"Tuna Sandwich\",\"type\":\"Food\"},
        {\"quantity\":0,\"id\":\"00000003\",\"name\":\"Salad\",\"type\":\"Fresh Food\"}]}";
```

GSON

```
Gson gson = new Gson();
Products products = gson.fromJson(jsonString, Products.class);

for(int i=0; i<products.products.size(); ++i){
    System.out.println(products.products.get(i).id);
    System.out.println(products.products.get(i).name);
    System.out.println(products.products.get(i).type);
    System.out.println(products.products.get(i).quantity);
}
```

YAML
Yet Another Markup Language

YAML

YAML Ain't Markup Language, a recursive acronym, to distinguish its purpose as data-oriented, rather than document markup.

YAML

YAML (/ˈjæməl/) *(see § History and name)* is a human-readable data serialization language. It is commonly used for configuration files and in applications where data is being stored or transmitted. YAML targets many of the same communications applications as Extensible Markup Language (XML) but has a minimal syntax which intentionally differs from Standard Generalized Markup Language (SGML).[3] It uses both Python-style indentation to indicate nesting, and a more compact format that uses `[...]` for lists and `{...}` for maps[3] but forbids tab characters to use as indentation[4] thus only some JSON files are valid YAML 1.2.

Custom data types are allowed, but YAML natively encodes scalars (such as strings, integers, and floats), lists, and associative arrays (also known as maps, dictionaries or hashes). These data types are based on the Perl programming language, though all commonly used high-level programming languages share very similar concepts.[6][7][8] The colon-centered syntax, used for expressing key-value pairs, is inspired by electronic mail headers as defined in RFC 822, and the document separator `---` is borrowed from MIME (RFC 2046). Escape sequences are reused from C, and whitespace wrapping for multi-line strings is inspired by HTML. Lists and hashes can contain nested lists and hashes, forming a tree structure; arbitrary graphs can be represented using YAML aliases (similar to XML in SOAP).[3] YAML is intended to be read and written in streams, a feature inspired by SAX.

https://en.wikipedia.org/wiki/YAML

YAML

- 2001-ben hozta létre:
  - Clark Evans
  - Ingy döt Net
  - Oren Ben-Kiki
- Széles körben a következőket szokták használni:

YAML

- Whitespace indentation is used for denoting structure; however, tab characters are not allowed as part of that indentation.
- Comments begin with the number sign (`#`), can start anywhere on a line and continue until the end of the line. Comments must be separated from other tokens by whitespace characters.[22] If # characters appear inside of a string, then they are number sign (`#`) literals.
- List members are denoted by a leading hyphen (`-`) with one member per line.
  - A list can also be specified by enclosing text in square brackets (`[...]`) with each entry separated by a comma.
- An associative array entry is represented using colon space in the form *key: value* with one entry per line. YAML requires the colon be followed by a space so that url-style strings like `http://www.wikipedia.org` can be represented without needing to be enclosed in quotes.
  - A question mark can be used in front of a key, in the form "?key: value" to allow the key to contain leading dashes, square brackets, etc., without quotes.
  - An associative array can also be specified by text enclosed in curly braces (`{...}`), with keys separated from values by colon and the entries separated by commas (spaces are not required to retain compatibility with JSON).
- Strings (one type of scalar in YAML) are ordinarily unquoted, but may be enclosed in double-quotes (`"`), or single-quotes (`'`).
  - Within double-quotes, special characters may be represented with C-style escape sequences starting with a backslash (`\`). According to the documentation the only octal escape supported is `\0`.
  - Within single quotes the only supported escape sequence is a doubled single quote (`''`) denoting the single quote itself as in `'don''t'`.
- Block scalars are delimited with indentation with optional modifiers to preserve (`|`) or fold (`>`) newlines.
- Multiple documents within a single stream are separated by three hyphens (`---`).
  - Three periods (`...`) optionally end a document within a stream.
- Repeated nodes are initially denoted by an ampersand (`&`) and thereafter referenced with an asterisk (`*`).
- Nodes may be labeled with a type or tag using a double exclamation mark (`!!`) followed by a string, which can be expanded into a URI.
- YAML documents in a stream may be preceded by 'directives' composed of a percent sign (`%`) followed by a name and space-delimited parameters. Two directives are defined in YAML 1.1:
  - The %YAML directive is used for identifying the version of YAML in a given document.
  - The %TAG directive is used as a shortcut for URI prefixes. These shortcuts may then be used in node type tags.

https://en.wikipedia.org/wiki/YAML#cite_note-yaml_org_about-17

YAML

```yaml
---
receipt:     Oz-Ware Purchase Invoice
date:        2012-08-06
customer:
    first_name:   Dorothy
    family_name:  Gale

items:
    - part_no:   A4786
      descrip:   Water Bucket (Filled)
      price:     1.47
      quantity:  4

    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      size:      8
      price:     133.7
      quantity:  1

bill-to:  &id001
    street: |
            123 Tornado Alley
            Suite 16
    city:   East Centerville
    state:  KS

ship-to:  *id001

specialDelivery:  >
    Follow the Yellow Brick
    Road to the Emerald City.
    Pay no attention to the
    man behind the curtain.
```

https://en.wikipedia.org/wiki/YAML