
Pengenalan JSON

Pengenalan JSON

- JSON singkatan dari JavaScript Object Notation
- Adalah salah satu format pertukaran data yang saat ini banyak digunakan
- Bentuk JSON mirip seperti Object di JavaScript
- <https://www.json.org/>

Tipe Data JSON

- JSON memiliki banyak tipe data, dan semuanya mengikuti tipe data di JavaScript
- Object
- Array
- Number
- String
- Boolean
- Bagaimana dengan tipe data lainnya seperti Date misal, maka akan menggunakan String atau Number, sesuai kesepakatan pembuatan data dan penerima data

Kode : Contoh JSON

```
1  {
2      "id": 1,
3      "married": true,
4      "name" : {
5          "first": "Budi",
6          "last": "Setiawan"
7      },
8      "age": 30,
9      "address": {
10         "street": "Jl. Jendral Sudirman",
11         "city": "Jakarta",
12         "country": "Indonesia"
13     },
14     "hobbies": [
15         "Reading",
16         "Swimming",
17         "Travelling"
18     ]
19 }
```

```
        "Travelling"
    ],
    "children": [
        {
            "name": "Andi",
            "age": 5
        },
        {
            "name": "Budi",
            "age": 3
        }
    ]
}
```

Pengenalan Jackson

JSON Library

- Saat dibuatnya video ini, Java tidak memiliki fitur bawaan untuk membuat data JSON
- Oleh karena itu, kita perlu menggunakan library untuk membuat dan membaca data JSON
- Ada banyak sekali library untuk JSON, seperti Jackson, GSON, FastJSON, dan lain-lain
- Pada kelas ini, kita akan menggunakan Jackson sebagai library untuk JSON nya



Pengenalan Jackson

- Jackson adalah salah satu library yang banyak digunakan oleh programmer Java terutama Backend untuk mengolah data JSON
- Jackson dikenal dengan kecepatannya memproses data JSON
- <https://github.com/FasterXML/jackson-databind/>

Membuat Project

Membuat Project

- <https://start.spring.io/>



Menambah Dependency

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.14.2</version>
</dependency>
<dependency>
```

Object Mapper

Object Mapper

- ObjectMapper adalah class di library Jackson sebagai class utama
- Untuk menggunakan library Jackson, kita perlu membuat object ObjectMapper
- ObjectMapper hanya perlu dibuat sekali, dan bisa digunakan berkali-kali, tanpa harus takut dengan race condition, karena sudah thread safe



Kode : Membuat Object Mapper

```
no usages new *  
  
@Test  
void createObjectMapper() {  
    ObjectMapper objectMapper = new ObjectMapper();  
  
    Assertions.assertNotNull(objectMapper);  
}  
}
```

Membuat JSON Object

JSON Object di Java

- JSON Object adalah data berisikan atribut dan value, dimana di Java hal ini mirip dengan tipe data Map
- Untuk membuat JSON, di Java kita bisa membuat dalam Map, lalu kita bisa menggunakan ObjectMapper untuk melakukan konversi ke JSON

Membuat JSON Object

- Untuk membuat JSON Object, kita bisa menggunakan method :
- `writeValue(output, object)`, dimana output bisa Writer, File, OutputStream
- `writeValueAsString(object)`, dimana hasilnya adalah JSON dalam String
- `writeValueAsBytes(object)`, dimana hasilnya adalah JSON dalam byte[]



Kode : Membuat JSON

```
@Test
void createJSON() throws JsonProcessingException {
    Map<String, Object> person = Map.of(
        "firstName", "Budi",
        "lastName", "Nugraha",
        "age", 30,
        "married", true
    );

    ObjectMapper objectMapper = new ObjectMapper();
    String json = objectMapper.writeValueAsString(person);

    System.out.println(json);    You, Moments ago • Uncommitted changes
```

Membaca JSON Object

Membaca JSON Object

- ObjectMapper selain digunakan untuk membuat JSON Object dari Map, bisa juga digunakan untuk melakukan kebalikannya, membaca JSON menjadi Map
- Kita bisa menggunakan method :
- `readValue(input, typereference)`, dimana input adalah InputStream, Reader, String, File, dan lain-lain. Dan typereference adalah class untuk Generic



Kode : Membaca JSON

```
@Test
void readJSON() throws JsonProcessingException {
    String json = """
        |{"age":30,"married":true,"firstName":"Budi","lastName":"Nugraha"}
        """;

    ObjectMapper objectMapper = new ObjectMapper();
    new *
    Map<String, Object> person = objectMapper.readValue(json, new TypeReference<Map<String, Object>>() {
});

    Assertions.assertEquals("Budi", person.get("firstName"));
    Assertions.assertEquals("Nugraha", person.get("lastName"));
}
```

Membuat JSON Array

Membuat JSON Array

- Untuk representasi JSON Array di Java, kita bisa menggunakan tipe data collection, seperti List atau Set misalnya
- Dan untuk membuat JSON Array dari tipe data collection, kita bisa menggunakan method yang sama seperti membuat JSON Object



Kode : Membuat JSON Array

```
@Test  
void createJsonArray() throws JsonProcessingException {  
    List<String> hobbies = List.of("Coding", "Reading", "Traveling");  
  
    ObjectMapper objectMapper = new ObjectMapper();  
    String json = objectMapper.writeValueAsString(hobbies);  
  
    System.out.println(json);  
}  
}
```

Membaca JSON Array

Membaca JSON Array

- Untuk membaca JSON Array, kita juga bisa menggunakan method yang sama seperti membaca JSON Object



Kode : Membaca JSON Array

```
@Test
void readJsonArray() throws JsonProcessingException {
    String json = """
        ["Coding", "Reading", "Traveling"]
        """;
}

ObjectMapper objectMapper = new ObjectMapper();
new *
List<String> hobbies = objectMapper.readValue(json, new TypeReference<List<String>>() {});
Assertions.assertEquals(List.of("Coding", "Reading", "Traveling"), hobbies);
}
```

Konversi Object ke JSON

Konversi Object ke JSON

- Salah satu kekurangan menggunakan Map ketika membuat JSON Object adalah, kita harus melakukan input data secara manual satu per satu ke dalam Map nya
- Di Java, biasanya tipe data akan dibuat dalam bentuk class Java Bean (getter setter)
- Jackson juga bisa otomatis melakukan konversi Object menjadi JSON secara otomatis, dimana attribute JSON nya akan menggunakan nama field di class nya
- Jika Object tersebut memiliki field dengan tipe data Object lain, Jackson juga akan otomatis membuat embedded Object atau Array secara otomatis

Kode : Person Class

```
no usages new *
3  public class Address {
4
5      2 usages
6      private String street;
7
8      2 usages
9      private String city;
10
11     2 usages
12     private String country;
```

```
no usages new
5  public class Person {
6
7      2 usages
8      private String id;
9
10     2 usages
11     private String name;
12
13     2 usages
14     private List<String> hobbies;
```



Kode : Konversi Object ke JSON

```
@Test
void createJsonFromObject() throws JsonProcessingException {
    Person person = new Person();
    person.setId("1");
    person.setName("Eko");
    person.setHobbies(List.of("Coding", "Reading"));

    Address address = new Address();
    address.setStreet("Jalan Belum Jadi");
    address.setCity("Jakarta");
    address.setCountry("Indonesia");
    person.setAddress(address);

    ObjectMapper objectMapper = new ObjectMapper();
    String json = objectMapper.writeValueAsString(person);
    System.out.println(json);
```

Konversi JSON ke Object

Konversi JSON ke Object

- Untuk melakukan konversi JSON ke dalam object class Java Bean, kita bisa melakukan dengan cara hal yang sama ketika melakukan konversi ke Map atau List
- Hanya saja, parameternya tidak perlu menggunakan TypeReference lagi, kita bisa langsung sebutkan nama class Java Bean nya



Kode : Konversi JSON ke Object

```
@Test
void createObjectFromJson() throws JsonProcessingException {
    String json = """
        {"id":"1","name":"Eko","hobbies":["Coding","Reading"],"address":{"street":"Jalan Belum Jadi","city":"
        """;
    }

    ObjectMapper objectMapper = new ObjectMapper();
    Person person = objectMapper.readValue(json, Person.class);

    Assertions.assertEquals("1", person.getId());
    Assertions.assertEquals("Eko", person.getName());
    Assertions.assertEquals("Jalan Belum Jadi", person.getAddress().getStreet());
    Assertions.assertEquals("Jakarta", person.getAddress().getCity());
    Assertions.assertEquals("Indonesia", person.getAddress().getCountry());
    Assertions.assertEquals(List.of("Coding", "Reading"), person.getHobbies());
}
```

Mapper Feature



Mapper Feature

- Saat kita membuat object dari class ObjectMapper, terdapat konfigurasi yang bisa kita lakukan pada ObjectMapper tersebut
- Kita bisa melihat daftar konfigurasi yang tersedia di ObjectMapper di halaman :
- <https://github.com/FasterXML/jackson-databind/wiki/Mapper-Features>



Kode : Mapper Feature

```
@Test
void mapperFeature() throws JsonProcessingException {
    ObjectMapper objectMapper = new ObjectMapper()
        .configure(MapperFeature.ACCEPT_CASE_INSENSITIVE_PROPERTIES, true);

    String json = """
        {"ID":"1","Name":"Eko"}
    """;

    Person person = objectMapper.readValue(json, Person.class);
    Assertions.assertEquals("1", person.getId());
    Assertions.assertEquals("Eko", person.getName());
}
```

Deserialization Feature



Deserialization Feature

- Jackson memiliki fitur yang bisa diaktifkan atau dinonaktifkan untuk proses deserialization (membaca JSON menjadi Java Object)
- Ada banyak sekali fitur yang terdapat di Jackson Deserialization, kita bisa lihat di link berikut :
- <https://github.com/FasterXML/jackson-databind/wiki/Deserialization-Features>



Kode : Deserialization Feature

```
@Test
void deserializationFeature() throws JsonProcessingException {
    ObjectMapper objectMapper = new ObjectMapper()
        .configure(DeserializationFeature.ACCEPT_SINGLE_VALUE_AS_ARRAY, true)
        .configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);

    String json = """
        |{"id":"1", "name":"Eko", "age":10, "hobbies":"Coding"}
        |""";
}

Person person = objectMapper.readValue(json, Person.class);
Assertions.assertEquals("1", person.getId());
Assertions.assertEquals("Eko", person.getName());
Assertions.assertEquals(List.of("Coding"), person.getHobbies());
}
```

Serialization Feature

Serialization Feature

- Jackson memiliki fitur yang bisa diaktifkan atau dinonaktifkan untuk proses serialization (menulis Java Object menjadi JSON)
- Ada banyak sekali fitur yang terdapat di Jackson Serialization, kita bisa lihat di link berikut :
- <https://github.com/FasterXML/jackson-databind/wiki/Serialization-Features>



Kode : Serialization Feature

```
@Test
void serializationFeature() throws JsonProcessingException {
    Person person = new Person();
    person.setId("1");
    person.setName("Eko");
    person.setHobbies(List.of("Coding", "Reading"));

    Address address = new Address();
    address.setStreet("Jalan Belum Jadi");
    address.setCity("Jakarta");
    address.setCountry("Indonesia");
    person setAddress(address);

    ObjectMapper objectMapper = new ObjectMapper()
        .configure(SerializationFeature.INDENT_OUTPUT, true);
    String json = objectMapper.writeValueAsString(person);
    System.out.println(json);
```

Serialization Inclusion



Serialization Inclusion

- Secara default, saat kita membuat JSON dari Java Object, semua field akan di-include, termasuk yang nilai field nya null
- Kita bisa mengubah cara Jackson melakukan include field mana saja yang akan dibuat sebagai attribute di JSON dengan menggunakan attribute Serialization Inclusion
- <https://fasterxml.github.io/jackson-annotations/javadoc/2.7/com/fasterxml/jackson/annotation/JsonInclude.Include.html>



Kode : Serialization Inclusion

```
@Test
void serializationInclusion() throws JsonProcessingException {
    ObjectMapper objectMapper = new ObjectMapper()
        .setSerializationInclusion(JsonInclude.Include.NON_NULL);

    Person person = new Person();
    person.setId("id");
    person.setName("Eko");

    String json = objectMapper.writeValueAsString(person);
    System.out.println(json);
}
```

Date Time

Date Time

- Saat kita membuat object Java, kita sering menambah field dengan tipe data Date Time, misal java.util.Date, java.util.Calendar, dan lain-lain
- JSON tidak memiliki tipe data Date Time
- Biasanya ketika menggunakan JSON, untuk menampilkan tipe data Date Time, kita memiliki beberapa pilihan, menampilkan dengan number (format milis) atau dengan string (misal format yyyy-MM-dd)
- Jackson secara default akan menampilkan Date Time dalam format number (milis)



Kode : Person Class

```
10 usages new *
6 public class Person {
7
8     2 usages
9         private String id;
0
1     2 usages
2         private Date createdAt;
3
4     2 usages
5         private Date updatedAt;
```



Kode : Date Time dalam Milis

```
@Test
void dateTimeInMilis() throws JsonProcessingException {
    ObjectMapper objectMapper = new ObjectMapper();

    Person person = new Person();
    person.setId("id");
    person.setName("Eko");
    person.setCreatedAt(new Date());
    person.setUpdatedAt(new Date());

    String json = objectMapper.writeValueAsString(person);
    System.out.println(json);
}
```

Date Time Non Milis

- Jika kita tidak ingin menggunakan milis ketika membuat atau membaca JSON, kita bisa gunakan juga string sebagai representasi Date Time nya
- Namun kita perlu memberitahu Date Format yang akan digunakan oleh Jackson
- Kita bisa ubah Date Formatter di ObjectMapper, dan menon-aktifkan fitur milis untuk Date Time



Kode : Date Time Formatter

```
@Test
void dateTimeFormatter() throws JsonProcessingException {
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    ObjectMapper objectMapper = new ObjectMapper()
        .configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false)
        .setDateFormat(dateFormat);

    Person person = new Person();
    person.setId("id");
    person.setName("Eko");
    person.setCreatedAt(new Date());
    person.setUpdatedAt(new Date());

    String json = objectMapper.writeValueAsString(person);
    System.out.println(json);
}
```

Jackson Annotation

Jackson Annotation

- Jackson secara default akan membuat JSON dari Java Class, dari membaca field yang ada, sampai membuat attribute di JSON nya
- Jackson menyediakan beberapa annotation yang bisa kita gunakan untuk mengubah behaviour default dari Jackson dengan cara menambahkan Jackson Annotation pada class Java nya
- <https://fasterxml.github.io/jackson-annotations/javadoc/2.7/com/fasterxml/jackson/annotation/package-summary.html>

Daftar Jackson Annotation

- `@JsonIgnore`, untuk menandai field yang akan di-ignore ketika proses serialization dan deserialization
- `@JsonFormat`, untuk mengubah format data (misal date time) ketika proses serialization dan deserialization
- `@JsonProperty`, untuk mengubah attribute ketika proses serialization dan deserialization



Kode : Class Person

```
16 usages new *
0 public class Person {
1
2     2 usages
3         private String id;
4
5     no usages
6         @JsonProperty("full_name")
7         private String fullName;
8
9     no usages
10        @JsonIgnore
11        private String password;
12
13     2 usages
14         private Date createdAt;
15
16     2 usages
17         @JsonFormat(pattern = "yyyy-MM-dd")
18         private Date updatedAt;
19
20 You, Moments ago • Uncommitted changes
```



Kode : Jackson Annotation

```
@Test
void annotation() throws JsonProcessingException {
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    ObjectMapper objectMapper = new ObjectMapper()
        .configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false)
        .setDateFormat(dateFormat);

    Person person = new Person();
    person.setId("id");
    person.setFullName("Eko");
    person.setPassword("rahasia");
    person.setCreatedAt(new Date());
    person.setUpdatedAt(new Date());

    String json = objectMapper.writeValueAsString(person);
    System.out.println(json);
```