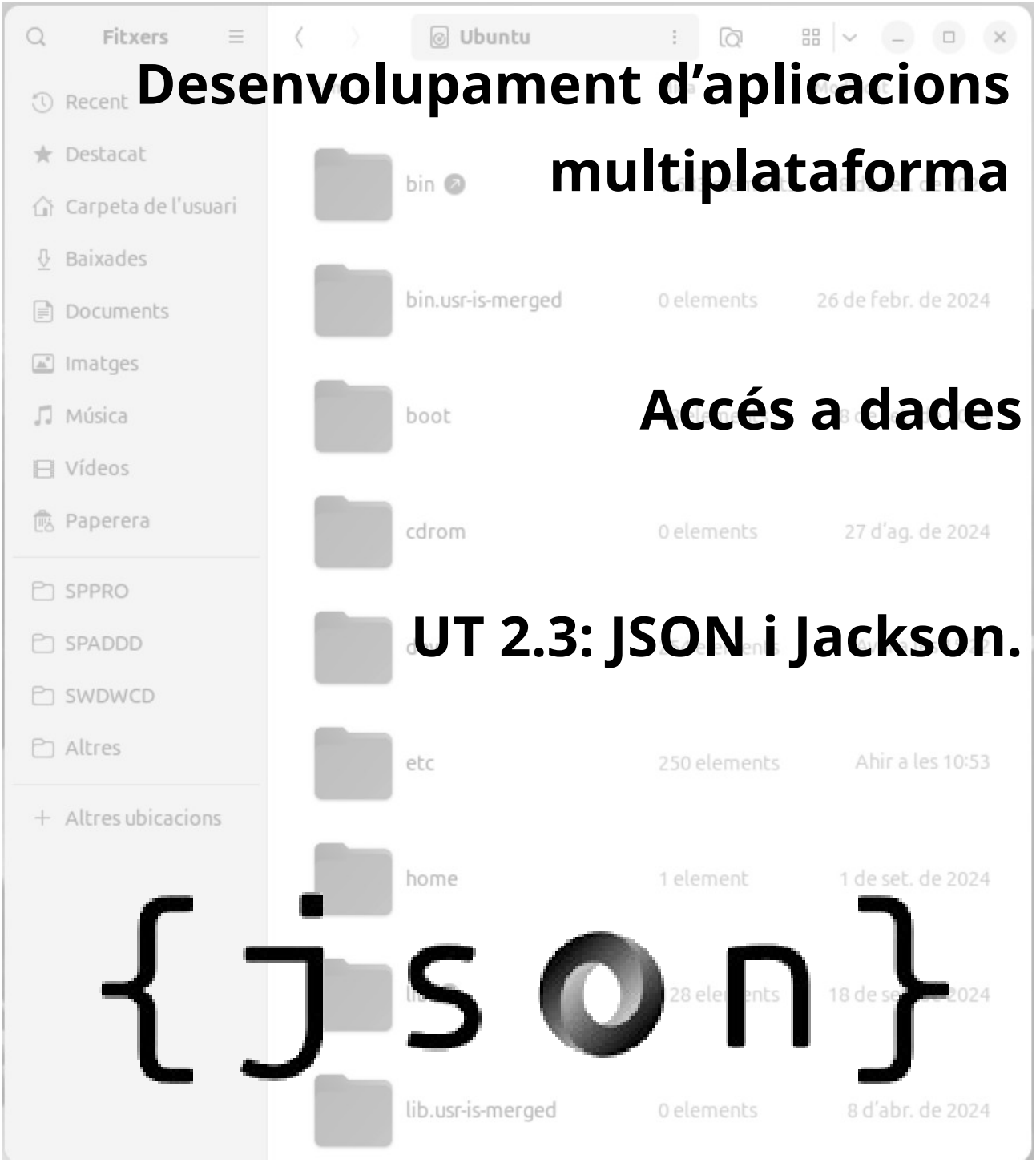
PAU CASESNOVES
Centre Integrat de Formació Professional



Desenvolupament d'aplicacions multiplataforma

Accés a dades

UT 2.3: JSON i Jackson.

Joan Pons Tugores

Índex de continguts

JSON....3
Valors....4
Objectes....5
Arrays....6
Conceptes relacionats amb la utilització de JSON....8
Binding....8
Marshalling o serialització....8
Unmarshalling o deserialització....8
POJO....8
Jackson....9
Mapatge d'un objecte Java a un document JSON....10
Reflection....10
Anotacions....10
ObjectMapper....11
Serialització o marshalling....11
Deserialització o unmarshalling....13
Serialització de llistes....14
Deserialització d'arrays JSON....14
Modificació del mapatge....16
Ignorar atributs....16
Ignorar atributs nulls....17
Canviar el nom....18
Canviar l'ordre de les propietats....19
Assignar àlies a una propietat....19

JSON

JavaScript Object Notation, llegit normalment com a “jaison” o “jotason”, **és un format d'intercanvi de dades** basat en una part de l'especificació d'ECMAScript, Javascript per els amics.

És un **format de text** completament independent del llenguatge, però que utilitza característiques i termes provinents dels llenguatges tipus C (C++, Java, Javascript, Python, ...)

Que sigui un format de text, com l'XML, implica que en utilitzar un JSON sempre es trobarà **dins d'una variable de tipus text o si és un literal, anirà tancat entre cometes**.

És una alternativa a XML. Com a característiques addicionals podem dir que és fàcil d'entendre i escriure per els humans i de generar i interpretar per les màquines. A més, la mateixa informació en format *json* ocupa molt menys espai que en format *XML*.

Un objecte amb JSON

```
‘{ “nom” : ”Joan”, “edat” : 4}’
```

L’equivalent amb XML:

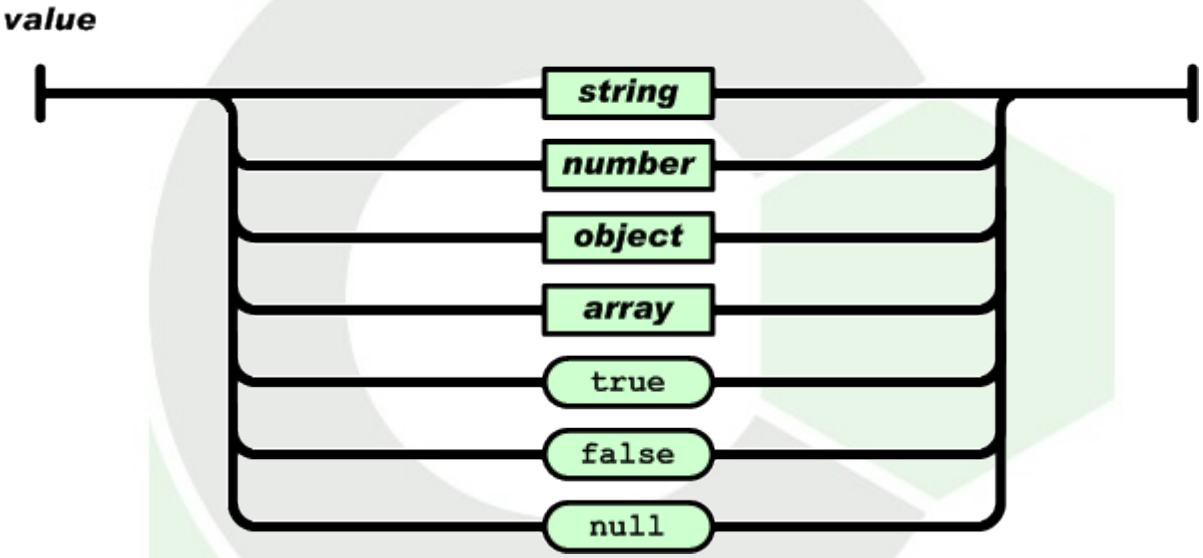
```
<persona> <nom>Joan</nom> <edat>4</edat> </persona>
```

JSON es basa en dues estructures freqüents a molts llenguatges de programació, objectes i arrays.

Valors

Els arrays poden contenir valors i els objectes parells nom – valor, però què és un valor?

Un **valor** pot ser una cadena de caràcters entre cometes dobles, o un número, o true o false, o null, o un objecte o un array.



Exemples de valors vàlids:

```
"Jo"
2345
12.05
true
false
null
{ “nom” : ”Joan” }
[ "jo", 12.05, null ]
```

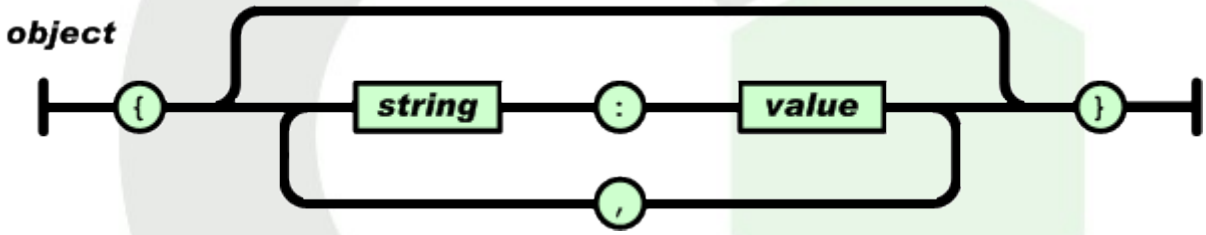
Els dos últims són, respectivament, un objecte i un array. Els veurem tot seguit.

Objectes

Objecte: Una col·lecció de parells nom - valor. És l'equivalent en altres llenguatges a objectes, registres, estructures, diccionaris, arrays associatius, ...

Un objecte comença amb *{* i acaba amb *}*. Cada nom d'atribut va entre **cometes dobles**, seguit per **:** i el valor. Els parells nom - valor estan separats per **,**

```
'{ "nom" : "Joan", "edat" : 4 }'
```



L'anterior diagrama ens diu que un objecte por tenir aquests tres formats, seguint els tres camins del diagrama:

1. Pot ser buit, si seguim el camí superior
2. Pot tenir un únic parell nom – valor si seguim el camí central
3. Pot tenir *n* parells nom – valor separats per comes si segueix el camí inferior després de definir cada parell.

Els següents exemples serien vàlids:

```
'{ }'
'{ "nom" : "Joan" }'
'{ "nom" : "joan", "edat" : 18, "adreça" : { "carrer" : "Joan Miró" , "numero" : 2,
"cp" : 07300, "localitat" : "Inca" } ,"telefons":[123123123, 321321321]}'
```

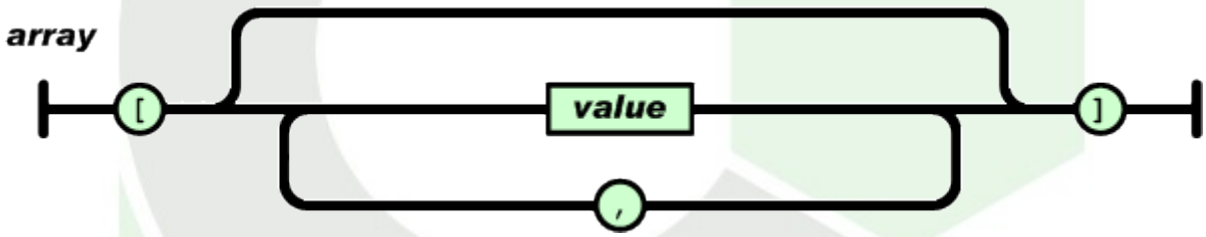
Com a valor podem tenir qualsevol dels valors descrits anteriorment, per tant un valor pot ser un objecte o un array també.

Arrays

Array: Una llista de valors. En la majoria dels llenguatges s'implementa com arrays, vectors, llistes o seqüències.

Un array comença amb *[* i acaba amb *]*. Els valor es separen amb ,

```
'[ "joan" , 4 , "blau" , true ]'
```



L'anterior diagrama ens diu que un array por tenir aquests tres formats, seguint els tres camins del diagrama:

1. Pot ser buit, si seguim el camí superior
2. Pot tenir un únic valor si seguim el camí central
3. Pot tenir *n* valor separats per comes si seguim el camí inferior després de cada valor.

Els següents exemples serien vàlids:

```
‘[ ]’
‘[ 12 ]’
‘[ 12, "asd", 234.75 ]’
‘[ 12, { “nom” : ”joan”, “edat” : 18 } ]’
‘[ 12, [ "asd", 234.75 ], true ]’
```

Conceptes relacionats amb la utilització de JSON

Binding

El Binding és una tècnica que consisteix a vincular classes Java amb formats específics d'emmagatzematge de manera automatitzada.

Depenent de la llibreria utilitzada de vegades necessitam especificar la relació entre les dades del document JSON i els atributs i les classes de la nostra aplicació, el que es diu *mapar*: El valor de l'atribut *name* del json s'assigna a la propietat *nom* de la classe *Alumne*.

Marshalling o serialització

El procés d'obtenir el document JSON (o XML, ...) a partir d'objectes Java es coneix amb el nom de *marshalling*.

Unmarshalling o deserialització

És el procés d'obtenir objectes Java a partir de documents JSON (o XML, ...)

POJO

Es refereix a una classe Java que s'utilitza per recollir o enviar dades des de l'aplicació a una font externa, per exemple una base de dades o un stream que utilitzi JSON o XML

Són classes que normalment tenen les següents característiques:

- Només tenen les propietats necessàries, amb els seus getters i a vegades els setters.
- Tenen el constructor sense paràmetres
- Solen tenir un constructor amb paràmetres per a tots els atributs.

Jackson

És una llibreria *open source* que inclou eines per transformar objectes Java a JSON i d'altres llenguatges d'intercanvi de dades com XML, i també permet fer el pas contrari, transformar documents en algun d'aquests llenguatges a objectes Java.

Per utilitzar aquesta llibreria al nostre projecte hem d'incloure els .jar amb les seves classes. Ho podem fer de diverses formes:

Si al nostre projecte utilitzam *maven* podem incloure la següent dependència:

```
<dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.17.2</version>
</dependency>
```

S'hauria de mirar quina és la darrera versió en el moment d'aplicar-la.

Aquesta dependència inclourà tres fitxers jar al nostre projecte:

- jackson-databind
- jackson-core
- jackson-annotations

Si el nostre projecte no utilitza *maven* haurem d'afegir aquests jar manualment. La manera de fer-ho és diferent a cada IDE, per tant ho hauríeu de mirar. A l'apartat *Annex* de l'aula del curs teniu com fer-ho al document *6.Paquets i jar.pdf* Teniu instruccions per als IDE's Eclipse, Netbeans, Visual Studio Code i Intellij.

Mapatge d'un objecte Java a un document JSON

El mapatge ja hem dit que consisteix en relacionar les propietats d'una classe Java amb les propietats definides a un document JSON o XML. Ho podem fer bàsicament de dues formes diferents: Amb Reflection i amb anotacions.

Reflection

És una característica del llenguatge Java que a partir d'una classe ens permet obtenir els noms dels seus membres, mètodes i atributs i manipular-los.

Afortunadament ho farà *jackson* per noltros per tant no ens farà falta dominar aquesta característica. jackson cercarà coincidències entre els noms de les propietats del document i els atributs de la classe. Crearà un objecte de la classe i assignarà valors als atributs segons aquestes coincidències. Per anar bé haurien de coincidir totes les propietat de la classe i del document.

Anotacions

Les anotacions són una característica del llenguatge Java que permet afegir informació al codi per a ser utilitzada per el compilador o per llibreries, ...

Segur que vos sona l'anotació *@Override* del *toString*. Aquesta anotació avisa al compilador que el mètode que decora, el mètode sobre el que apareix, en sobreescriu un de la superclasse i per tant la seva signatura ha de coincidir exactament amb la del mètode que sobreescriu.

Jackson utilitza anotacions per modificar el mapatge que generaria amb reflection. Per exemple, si tenim al codi

```
@JsonIgnore
private String llinatges;
```

jackson no inclourà l'atribut llinatges al mapatge.

ObjectMapper

És la classe de la llibreria encarregada de fer el marshaling i l'unmarshalling, és a dir, d'obtenir el document JSON a partir de l'objecte Java i al revés, d'obtenir l'objecte Java a partir del document JSON.

Per exemple, a la nostra aplicació podem utilitzar la classe *Autor*:

```
public class Autor {
  private String nom;
  private String Llinatges;
  //Constructors, getters i setters
  //...
}
```

I cream un objecte d'aquesta classe:

```
Autor ramon = new Autor("Ramón", "Llull");
```

Amb l'*ObjectMapper* podem obtenir el json d'aquest objecte i al revés, donat un json podem obtenir l'objecte Java.

Serialització o marshalling

Podem obtenir el JSON d'aquest objecte amb el mètode writeValue()

```
ObjectMapper objectMapper=new ObjectMapper();
objectMapper.writeValue(System.out, ramon);
```

El codi anterior hauria de mostrar les dades de l'objecte en format JSON:

```
{"nom":"Ramón","llinatges":"Llull"}
```

Podem posar-lo "guapo", però només té sentit si l'ha de llegir un humà, sinó no convé ja que augmenta la mida del json i això sempre és una mala idea.

```
objectMapper
        .writerWithDefaultPrettyPrinter()
        .writeValue(System.out ,ramon);
```

Ens generarà el següent JSON:

```
{
  "nom" : "Ramón",
  "llinatges" : "Llull"
}
```

A efectes pràctic no és massa útil. Però si tenim en compte que *System.out* és un *Stream*, ho podem substituir per un *OutputStream* per guardar les dades a un fitxer, per exemple, podem utilitzar un BufferedWriter ja que es tracta de generar un fitxer de text.

```
objectManager.writeValue(new BufferedWriter(new FileWriter(ruta)),ramon);
```

Ens generarà el fitxer que determini el paràmetre *ruta* i el seu contingut serà el JSON que representa l'objecte *ramon*.

Si ens interessa guardar el jason dins una variable podem utilitzar els mètodes:

- .writeValueAsString que torna el json dins d'una cadena de caràcters.
- .writeValueAsBytes que torna el json com un array de bytes.

Deserialització o unmarshalling

Si necessitam incorporar a la nostra aplicació les dades que ens arriben en format JSON haurem d¡utilitzar alguna de les sobrecàrregues del mètode *readValue*. Per exemple, si volem recuperar les dades que hem guardat al fitxer de l'exemple anterior:

```
Autor recuperat = objectManager.readValue(
        new BufferedReader(new FileReader(ruta)),
        Autor.class);
```

El primer paràmetre és la font del JSON, en aquest cas un *stream*.

El segon paràmetre és el tipus de l'objecte que representa el json. La classe ha de tenir els mateixos atributs que té el json. Si al json apareix un atribut que no existeix a la classe es llençarà una excepció del tipus *DataBindException*. En canvi si la classe té atributs que no apareixen al JSON no passa res.

El mètode torna un objecte de la classe que hem posat al segon paràmetre.

El mètode *readValue* ens permet llegir JSON des de:

- una cadena de text
- un array de bytes
- una URL, representada per un objecte de la classe URL.
- un stream de tipus reader.
- ...

Per exemple, per recuperar les dades des d'una API remota:

```
URI uri=new
        URI("http://daw.paucasesnovescifp.cat:8080/Biblioteca/autors/1");
URL urlRamon=l.toURL();
Autor remot=om.readValue(urlRamon, Autor.class);
```

Serialització de llistes

Les llistes es serialitzen com a arrays en JSON. Per exemple si tenim un ArrayList<Autor> amb tres autors el seu JSON serà:

```
[
        {"nom":"Ramón","llinatges":"Llull"},
        {"nom":"Joanot","llinatges":"Martorell"},
        {"nom":"Miguel","llinatges":"Cervantes"}
]
```

Per exemple, per guardar aquest json en un fitxer

```
objectManager.writeValue(new BufferedWriter(new FileWriter(ruta)),autors);
```

Deserialització d'arrays JSON

Si tenim un json amb un array com el del punt anterior hem d'utilitzar la classe *TypeReference* per especificar en quin tipus volem convertir l'array.

Per exemple, si volem convertir l'array en un conjunt:

```
HashSet<Autor> recuperats = objectManager.readValue(
        new BufferedReader(new FileReader(ruta)),
        new TypeReference<HashSet<Autor>>() {});
```

En canvi, per obtenir un ArrayList<Autor>:

```
ArrayList<Autor> recuperats = objectManager.readValue(
        new BufferedReader(new FileReader(ruta)),
        new TypeReference<ArrayList<Autor>>() {});
```

Si volem un array:

```
Autor[] recuperats = objectManager.readValue(
        new BufferedReader(new FileReader(ruta)),
        Autor[].class);
```

Modificació del mapatge

A tots els exemples que hem posat fins ara s'han mapat tots els atributs de les classes java a propietats dels documents JSON amb el mateix nom.

En alguns casos ens pot interessar que això no sigui així: pot ser hi ha qualque atribut que no volem incloure al json, o que la propietat del document no tengui el mateix nom que l'atribut Java, o ...

Per a totes aquestes coses utilitzarem anotacions.

Ignorar atributs

Si decoram un atribut amb l'anotació *@JsonIgnore* aquest atribut no s'inclourà al json

```
public class Autor {
    private String nom;
    private String llinatges;

    @JsonIgnore
    private String nacionalitat;

    private Integer anyNeixement;
```

En aquest cas, com que l'atribut *nacionalitat* està decorat(té a sobre) l'anotació *@JsonIgnore*, el JSON no l'inclourà.

> **Important**: Recordau que les anotacions només afecten a l'element que tenen immediatament a sota. En aquest cas *anyNeixement* s'inclourà al json.

També podem decorar la classe amb l'anotació @JsonIgnoreProperties. Té com a paràmetre un array de cadenes amb els noms de les propietats que volem ignorar:

```
@JsonIgnoreProperties({"nacionalitat", "anyNeixement"})
public class Autor {
    private String nom;
    private String llinatges;
    private String nacionalitat;
    private Integer anyNeixement;
```

Ignorar atributs nulls

Podem especificar que no volem al json cap atribut que sigui null

Ho podem fer a nivell de classe, en aquest cas no s'inclourà al JSON cap atribut que sigui null:

```
@JsonInclude(JsonInclude.Include.NON_NULL)
public class Autor {
    private String nom;
    private String llinatges;
    private String nacionalitat;
    private Integer anyNeixement;
```

Per exemple si tenim el següent objecte:

```
Autor{nom='Joanot', llinatges='Martorell', nacionalitat='null',
                    anyNeixement=null}
```

Generarà el json:

```
{"nom":"Joanot","llinatges":"Martorell"}
```

També ho podem fer a nivell d'atribut, de manera que si tenim

```
public class Autor {
    private String nom;
    private String llinatges;
    @JsonInclude(JsonInclude.Include.NON_NULL)
    private String nacionalitat;
    private Integer anyNeixement;
```

Generarà el JSON:

```
{"nom":"Joanot","llinatges":"Martorell","anyNeixement":null}
```

Canviar el nom

Si volem que un atribut de la classe aparegui al json amb un altre nom hem de decorar aquest atribut amb *@JsonProperty("nomNou")*.

```
public class Autor {
    @JsonProperty("name")
    private String nom;
    private String llinatges;
    @JsonInclude(JsonInclude.Include.NON_NULL)
    private String nacionalitat;
    private Integer anyNeixement;
```

En aquest cas generarem el següent JSON:

```
{"llinatges":"Martorell","anyNeixement":null,"name":"Joanot"}
```

Canviar l'ordre de les propietats

Al darrer exemple la propietat *name* apareix al final del JSON. Si volem canviar l'ordre ho podem fer decorant la classe amb l'anotació *@JsonPropertyOrder*. Té com a paràmetre un array amb cadendes de caràcters amb els noms de les propietats en l'ordre en el que les volem:

```
@JsonPropertyOrder({"nom", "llinatges", "nacionalitat","anyNeixement"})
public class Autor {
    private String nom;
    private String llinatges;
    private String nacionalitat;
    private Integer anyNeixement;
```

El JSON generat serà:

```
{"nom":"Joanot","llinatges":"Martorell","anyNeixement":null}
```

Assignar àlies a una propietat

Si ens pot passar que una propietat ens arribi amb distints noms podem utilitzar l'anotació *@JsonAlias* per especificar tots els altres noms que pot rebre aquesta propietat. Aquesta anotació té com a paràmetre un array de cadenes de caràcters amb els noms alternatius: Per exemple, si la propietat *llinatges* pot aparèixer també com a *cognoms* o *apellidos:*

```
public class Autor {
    private String nom;
    @JsonAlias({"cognoms", "apellidos"})
    private String llinatges;
```

Si al json ens arriba llinatges, cognoms o apellidos es maparà a l'atribut llinatges.