HTTP Protocol
&
Consuming Web APIs: GET

Anatomy of an IP Web Address/URL

Here is a web address (as a Universal Resource Locators or URL) using an **IP address**:

https://**127.0.0.1**:**3000**

- **protocol:** others - http, ftp
- **ip address:** This is the unique address of a machine on a network.
- **port**: Number allocated for a specific type of service.

Anatomy of a Named Host Web Address

Here is another URL that uses hostnames, this is certainly easier to remember than a bunch of numbers.

https://**skynet**.**wecomeinpeace.com**

- **host name**: A physical name assigned to your machine.
- **domain name**: Defines a specific “region of control” on the internet, also, .com is referred to as the top-level domain name.

The above URL is an example of a **fully qualified domain name**.

DNS & DNS Servers

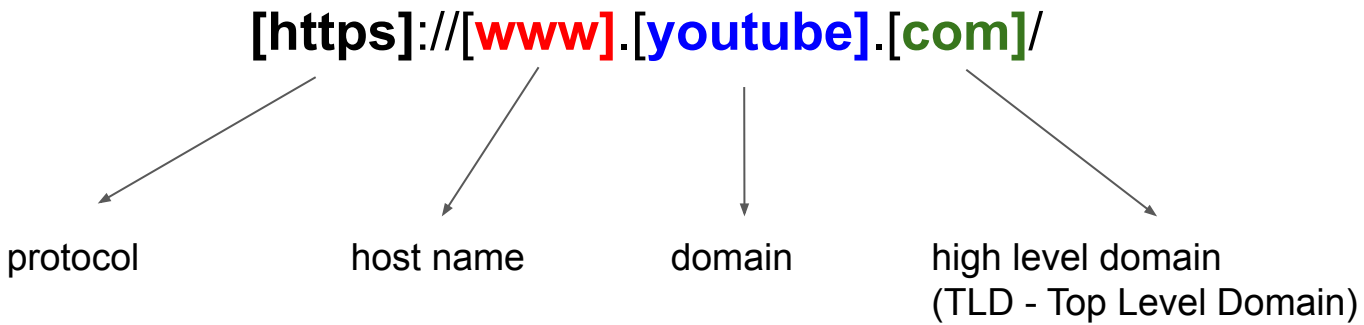
- **DNS** is an acronym for Domain Name System.

- A **DNS server** is responsible for converting a URL containing human readable domain names to one containing an IP address.
    - https://skynet.wecomeinpeace.com to https://127.0.0.1:3000

So ... www ?

Because I'm sure you've wondered…

- On a URL the appearance of www has no bearing on the means by which we are communicating with another machine on the network (the protocol is still http or https).
- www is simply a hostname:



What is an API ?

- **Application Programming Interface (API)**
- A set of functions and/or procedures designed to interact with an external system.
- Modern cloud architecture relies heavily on API's.
- **Consuming an API** means interacting with an API's code to produce a desired result.
- Most modern API's use the **Representational state transfer (REST)** model implemented via the **HTTP Protocol** (more on this next)
    - API's or services that use the REST model are said to be **RESTful Web Services**.

HTTP Protocol

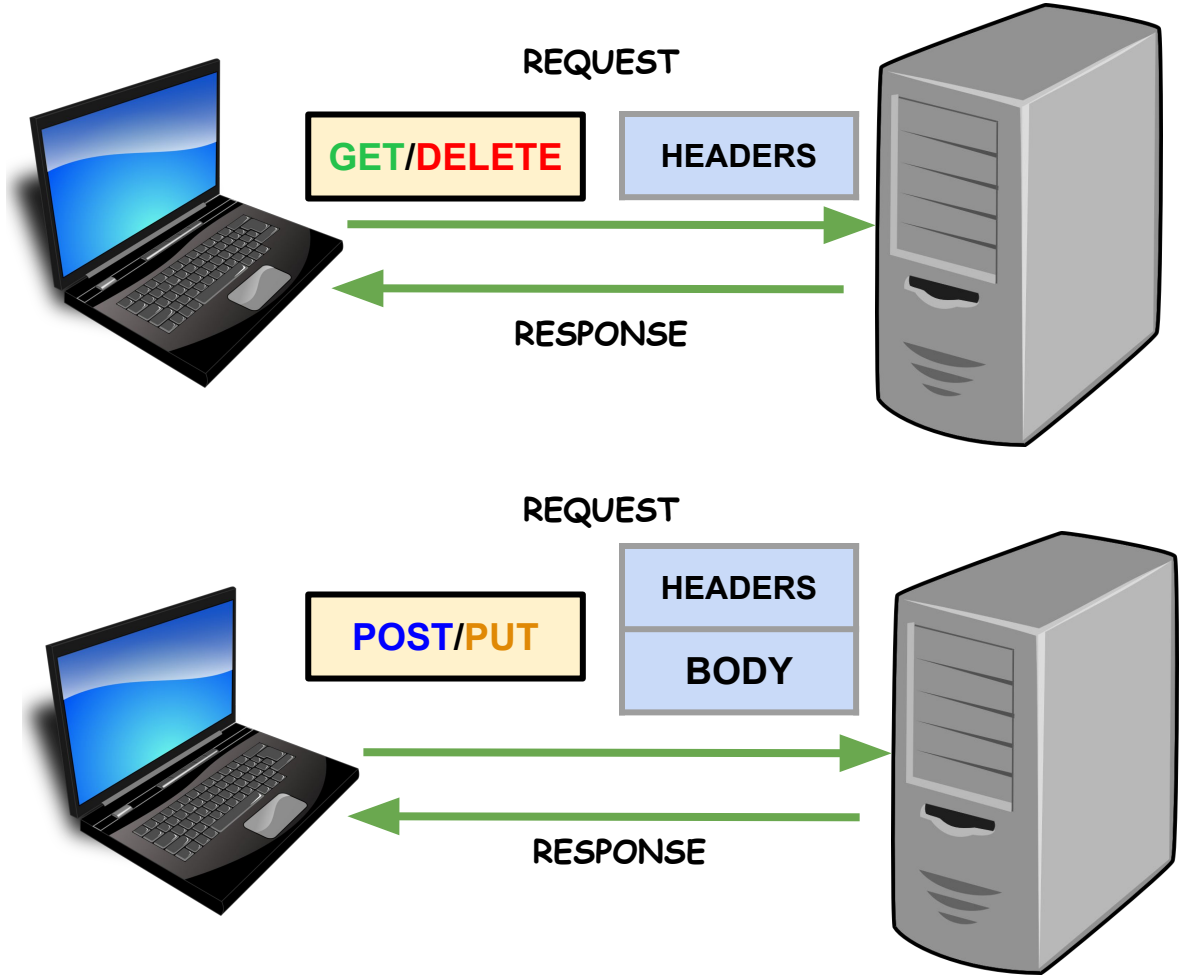
- Request/response protocol
  - Uses request and response messages
- Stateless
- Provides ability to implement authentication
- Resources access via **Universal Resource Locators (URLs)**
  - Implemented using the Uniform Resource Identifiers (URI's) schemes http and https.
- Defines methods to indicate the desired action to be performed on the identified resource.
  - We will use:
    - GET
    - POST
    - PUT
    - DELETE

GET:
Retrieve Resource

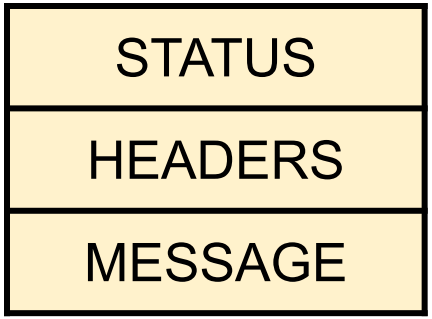
POST:
Create Resource

PUT:
Update Resource

DELETE:
Remove Resource



HTTP Response




```
HTTP/1.1 200 OK
```

```
Content-Type: application/json; charset=utf-8
```

```
[
    {
        "hotelID": 1,
        "title": "What a great hotel!",
        "review": "It was great!",
        "author": "John Smith",
        "stars": 4
    }
}
```

Possible Responses from HTTP Request

Once a request is made, the REST server can respond with specific status codes:

- **2xx**: All's well, the request was successful.
- **4XX**: The client (you or your application) has not structured the request correctly. Common examples of these are 400 Bad Request and 401 Unauthorized Request.
- **5XX**: The server has encountered some kind of error. The most common of these is the 500 Internal Server Error message.

Let's Try Making some API Calls Using Postman...

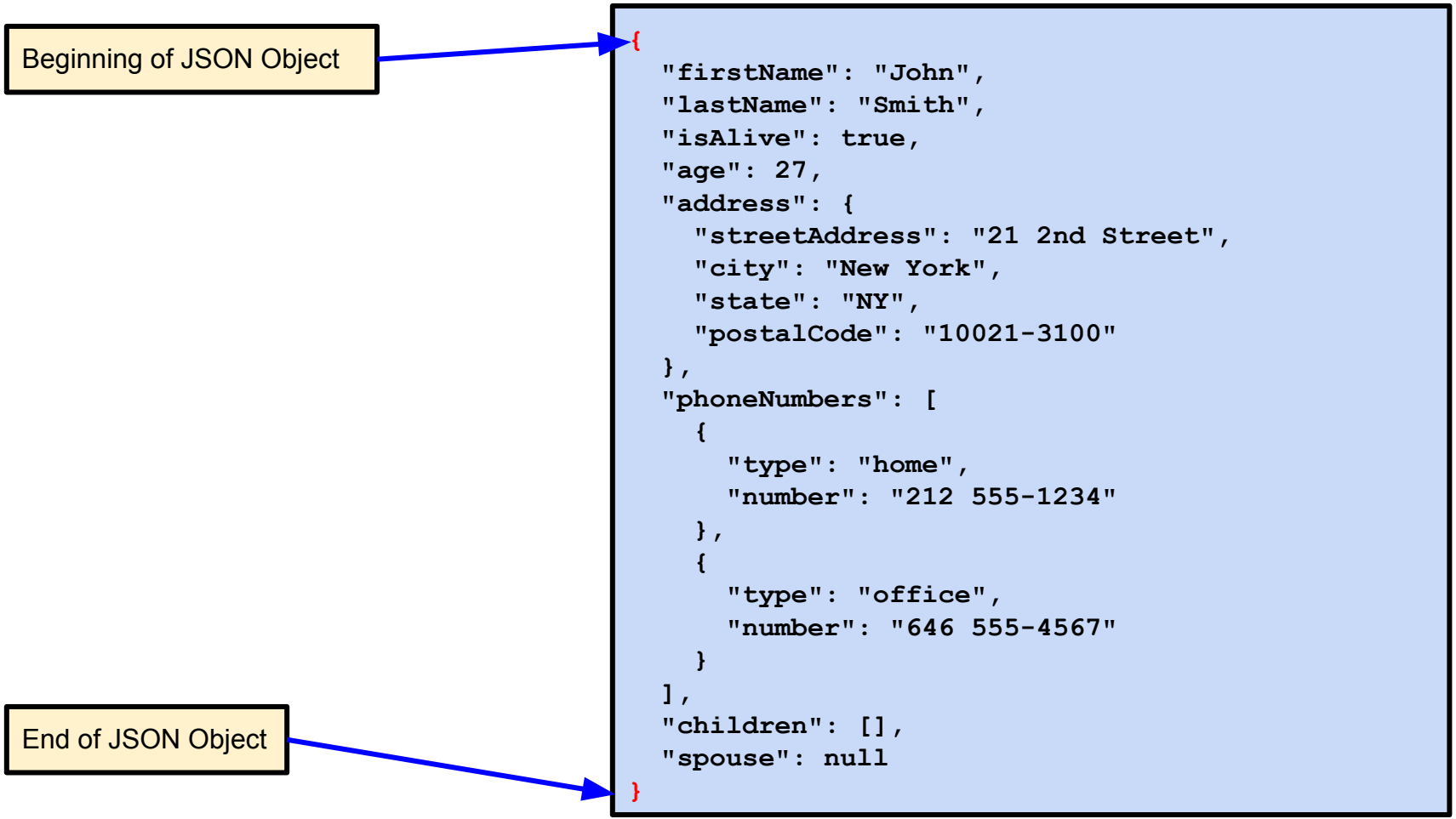
Let's Try a Public API Out on the Web...

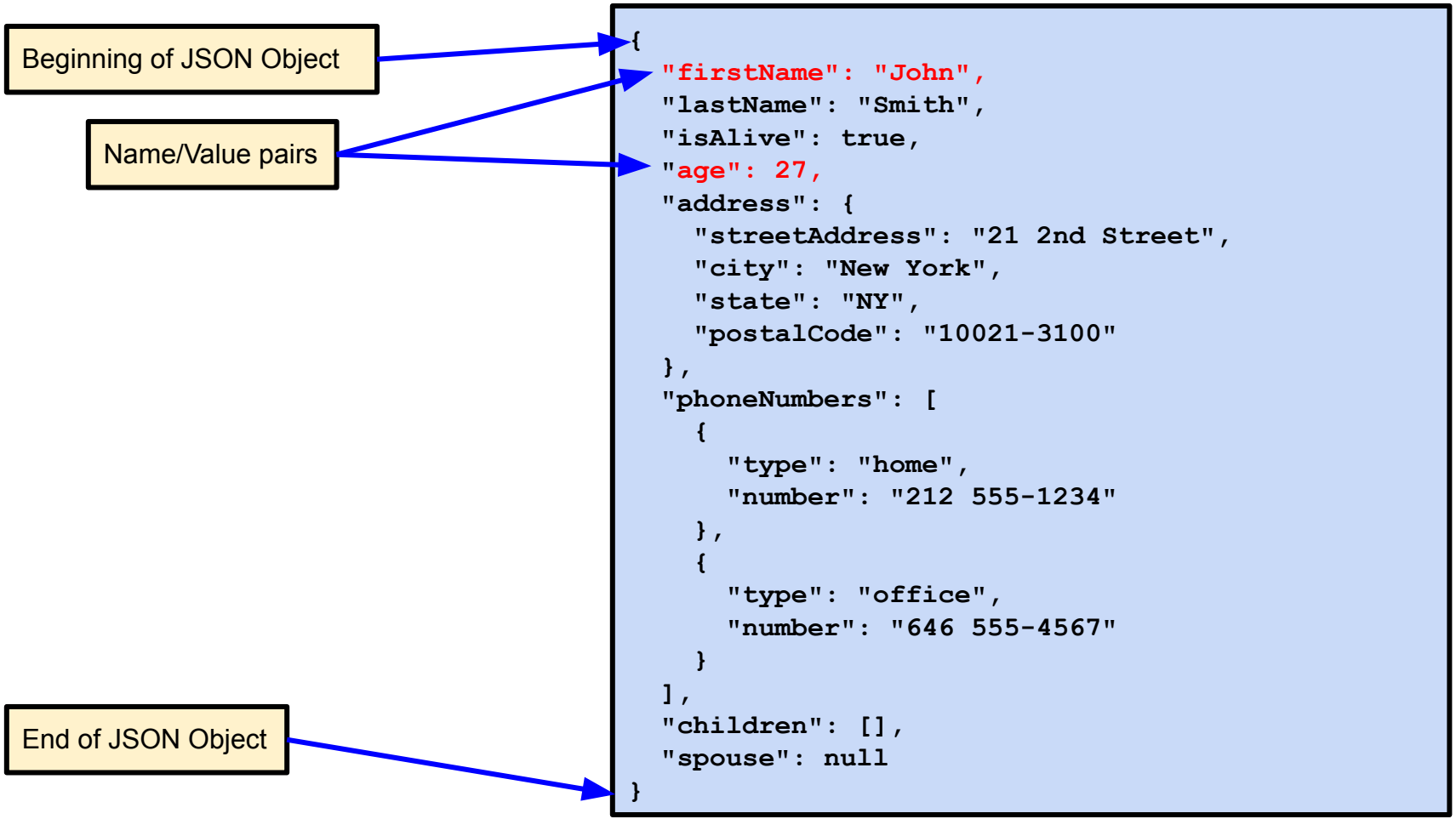
JavaScript Object Notation (JSON)

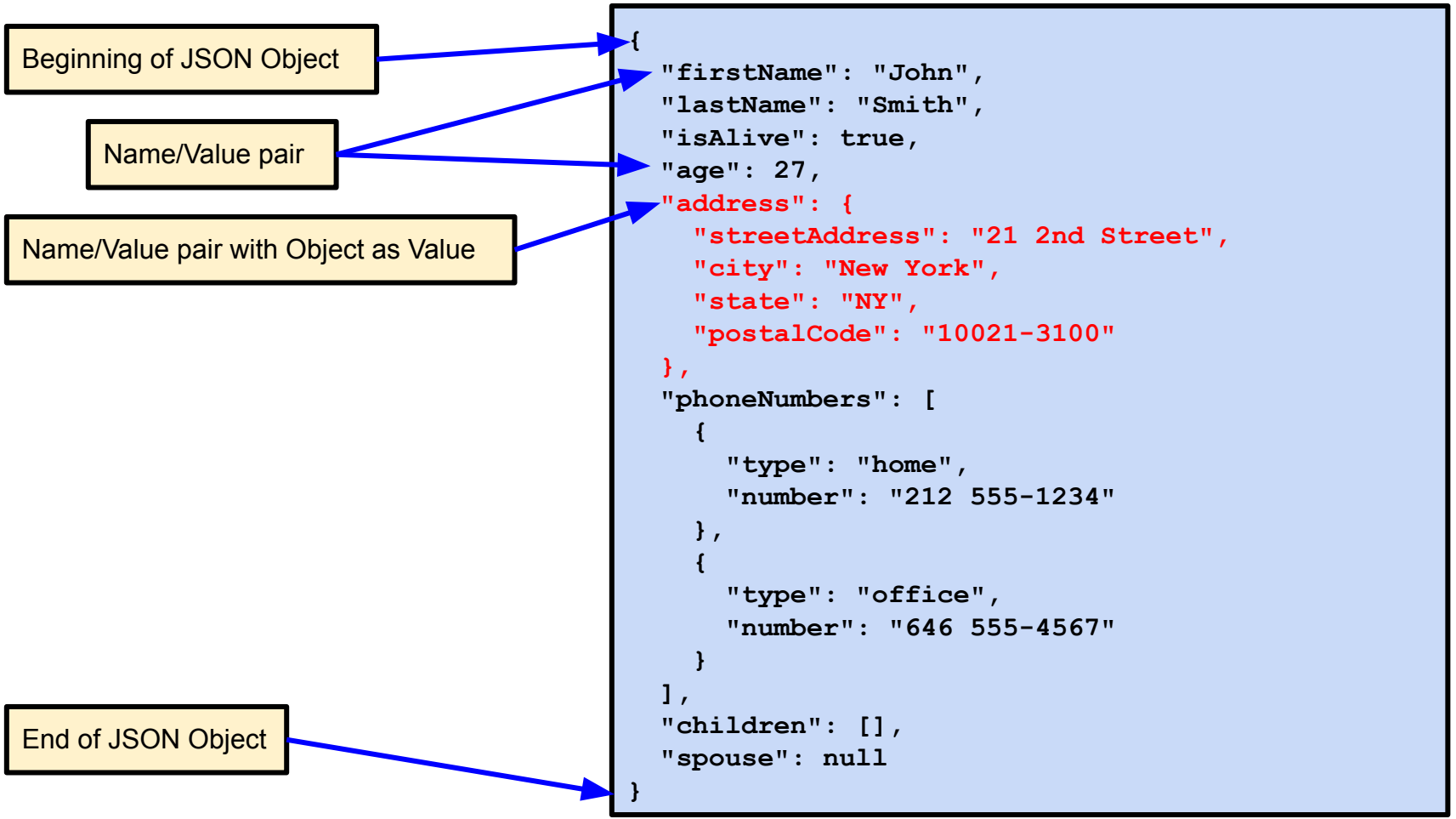
- Lightweight data-interchange format
- Easy for humans to read & write
- Easy to parse in code
- Text Only/Language Independent
- Often leveraged for passing data around the internet
- Made up of name/value pairs
- Name is always a quoted string (i.e. "address")
- Name/value separated by :
- Value can be one of several types

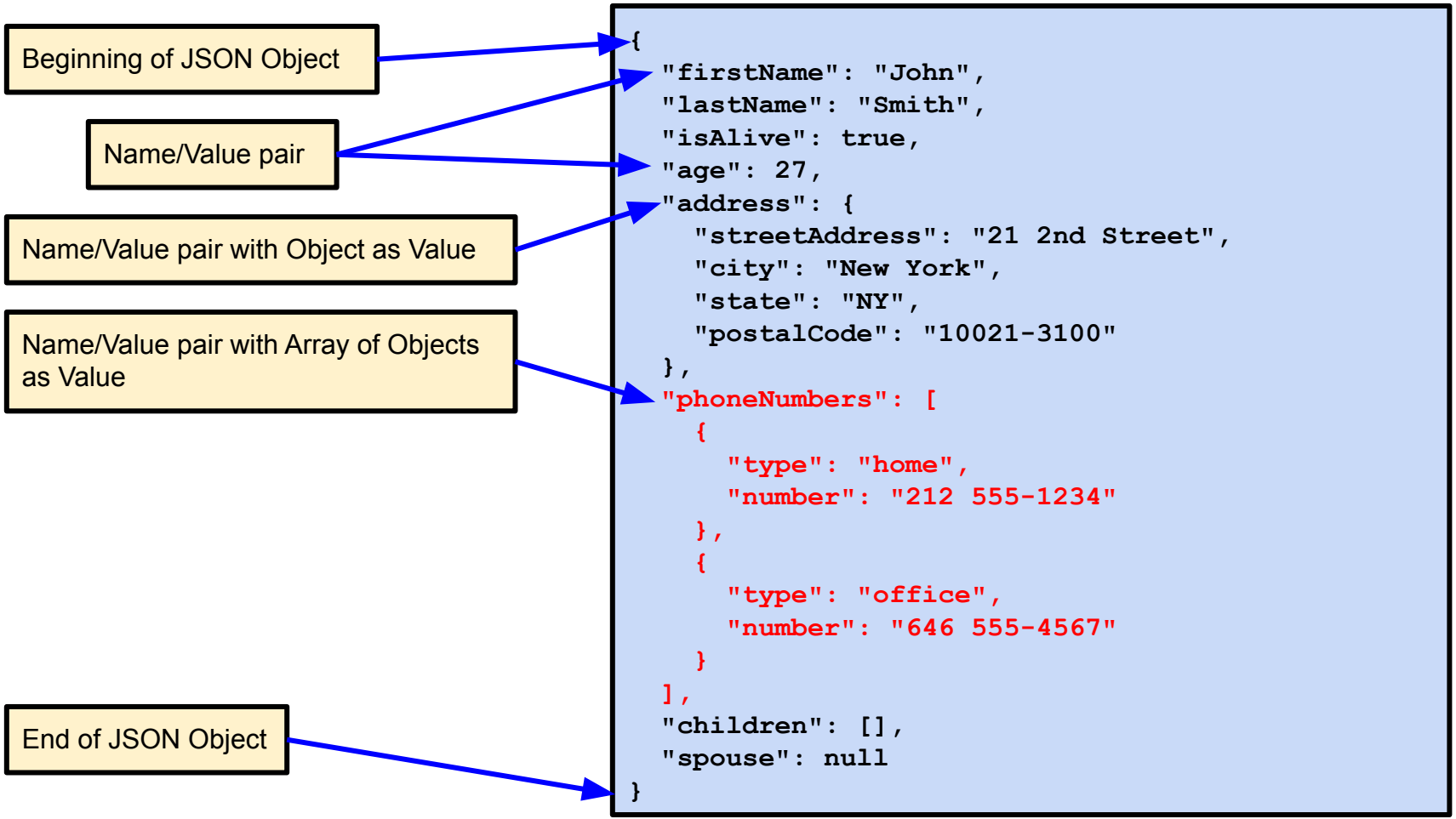
JSON Data Types

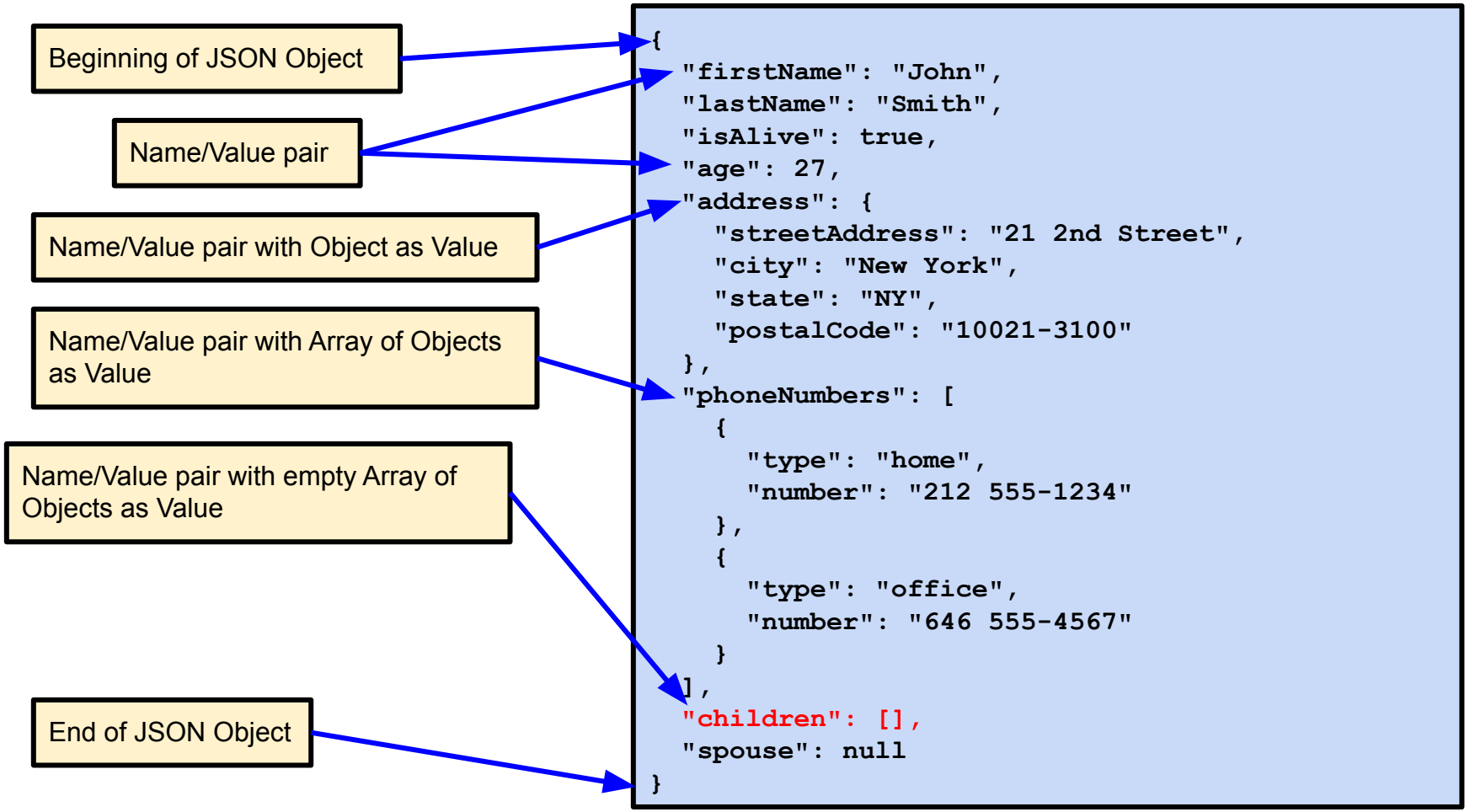
- **Number**
  - Any number (integer or decimal)
- **String**
  - A sequence of zero or more Unicode characters. Strings are delimited with double-quotation marks and support a backslash escaping syntax.
- **Boolean**
  - Either of the values `true` or `false`
- **Array**
  - An ordered list of zero or more values, each of which may be of any type. Arrays use `[]` square bracket notation with comma-separated elements.
- **Object**
  - A collection of name–value pairs where the names (also called keys) are strings. Objects are delimited with `{}` curly braces and use commas to separate each pair, while within each pair the colon ':' character separates the key or name from its value.
- `null`

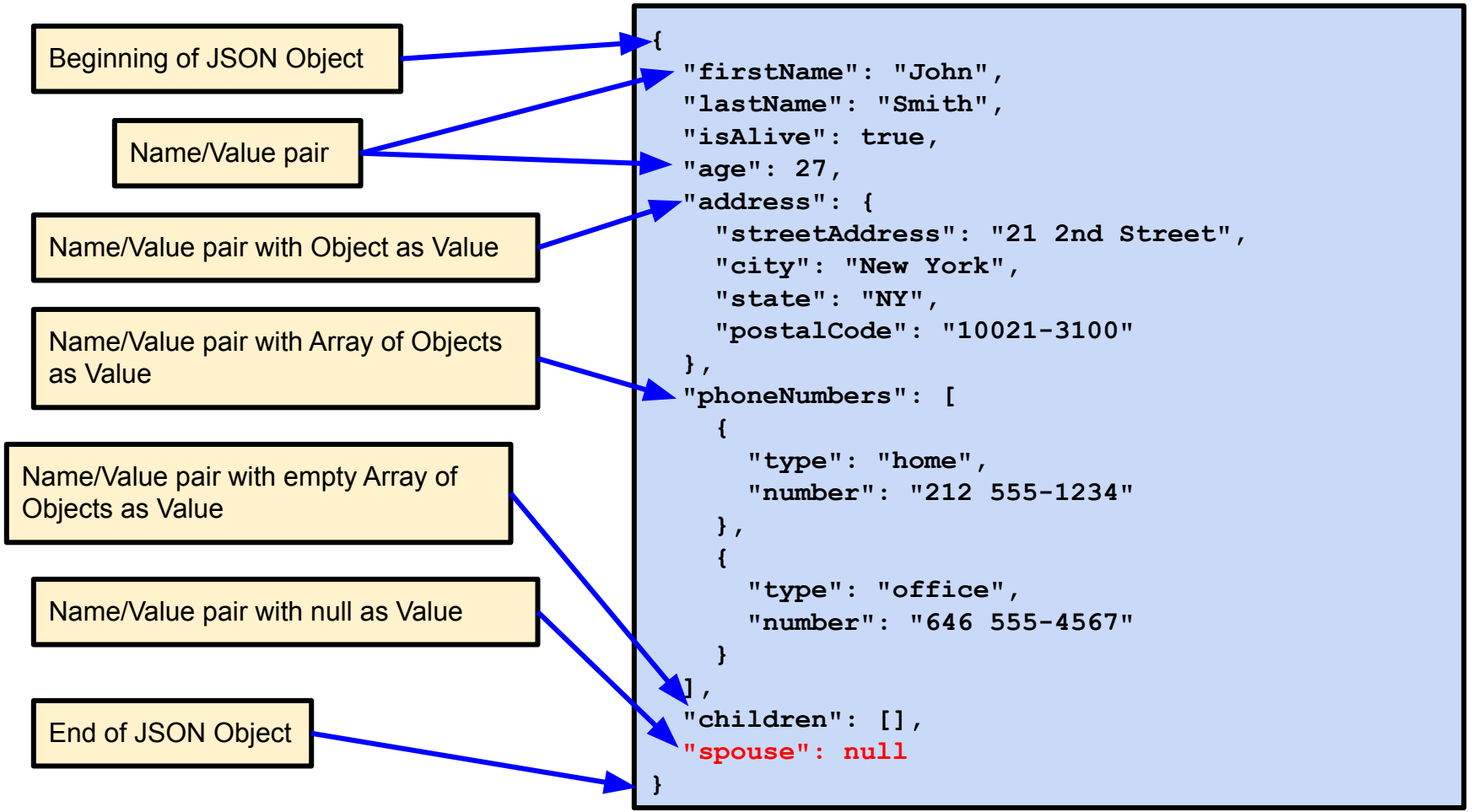

Beginning of JSON Object
End of JSON Object
{
"firstName": "John",
"lastName": "Smith",
"isAlive": true,
"age": 27,
"address": {
"streetAddress": "21 2nd Street",
"city": "New York",
"state": "NY",
"postalCode": "10021-3100"
},
"phoneNumbers": [
{
"type": "home",
"number": "212 555-1234"
},
{
"type": "office",
"number": "646 555-4567"
}
],
"children": [],
"spouse": null
}

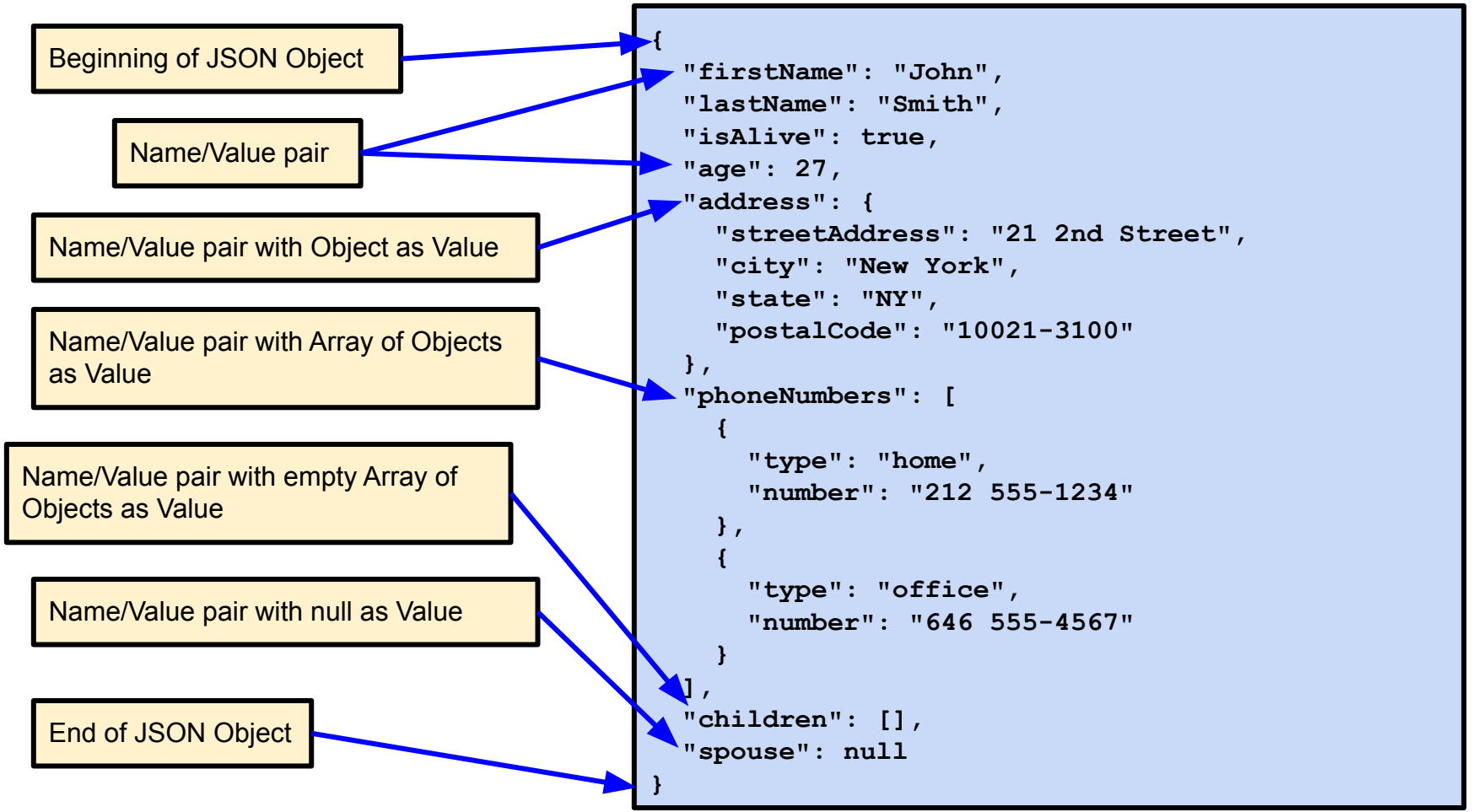

Beginning of JSON Object
Name/Value pairs
End of JSON Object
{
"firstName": "John",
"lastName": "Smith",
"isAlive": true,
"age": 27,
"address": {
"streetAddress": "21 2nd Street",
"city": "New York",
"state": "NY",
"postalCode": "10021-3100"
},
"phoneNumbers": [
{
"type": "home",
"number": "212 555-1234"
},
{
"type": "office",
"number": "646 555-4567"
}
],
"children": [],
"spouse": null
}


Beginning of JSON Object
Name/Value pair
Name/Value pair with Object as Value
End of JSON Object
{
"firstName": "John",
"lastName": "Smith",
"isAlive": true,
"age": 27,
"address": {
"streetAddress": "21 2nd Street",
"city": "New York",
"state": "NY",
"postalCode": "10021-3100"
},
"phoneNumbers": [
{
"type": "home",
"number": "212 555-1234"
},
{
"type": "office",
"number": "646 555-4567"
}
],
"children": [],
"spouse": null
}


Beginning of JSON Object
Name/Value pair
Name/Value pair with Object as Value
Name/Value pair with Array of Objects as Value
End of JSON Object
{
"firstName": "John",
"lastName": "Smith",
"isAlive": true,
"age": 27,
"address": {
"streetAddress": "21 2nd Street",
"city": "New York",
"state": "NY",
"postalCode": "10021-3100"
},
"phoneNumbers": [
{
"type": "home",
"number": "212 555-1234"
},
{
"type": "office",
"number": "646 555-4567"
}
],
"children": [],
"spouse": null
}


Beginning of JSON Object
Name/Value pair
Name/Value pair with Object as Value
Name/Value pair with Array of Objects as Value
Name/Value pair with empty Array of Objects as Value
End of JSON Object
{
"firstName": "John",
"lastName": "Smith",
"isAlive": true,
"age": 27,
"address": {
"streetAddress": "21 2nd Street",
"city": "New York",
"state": "NY",
"postalCode": "10021-3100"
},
"phoneNumbers": [
{
"type": "home",
"number": "212 555-1234"
},
{
"type": "office",
"number": "646 555-4567"
}
],
"children": [],
"spouse": null
}


Beginning of JSON Object
Name/Value pair
Name/Value pair with Object as Value
Name/Value pair with Array of Objects as Value
Name/Value pair with empty Array of Objects as Value
Name/Value pair with null as Value
End of JSON Object
{
"firstName": "John",
"lastName": "Smith",
"isAlive": true,
"age": 27,
"address": {
"streetAddress": "21 2nd Street",
"city": "New York",
"state": "NY",
"postalCode": "10021-3100"
},
"phoneNumbers": [
{
"type": "home",
"number": "212 555-1234"
},
{
"type": "office",
"number": "646 555-4567"
}
],
"children": [],
"spouse": null
}


Beginning of JSON Object
Name/Value pair
Name/Value pair with Object as Value
Name/Value pair with Array of Objects as Value
Name/Value pair with empty Array of Objects as Value
Name/Value pair with null as Value
End of JSON Object
{
"firstName": "John",
"lastName": "Smith",
"isAlive": true,
"age": 27,
"address": {
"streetAddress": "21 2nd Street",
"city": "New York",
"state": "NY",
"postalCode": "10021-3100"
},
"phoneNumbers": [
{
"type": "home",
"number": "212 555-1234"
},
{
"type": "office",
"number": "646 555-4567"
}
],
"children": [],
"spouse": null
}

Serialization & Deserialization

Java Objects can easily be converted to JSON data and vice versa, Mapping from an Object to JSON is known as **serialization** of an Object. Mapping from JSON to an Object is known as **deserialization**.

Java

```
Review[];

public class Review {
    private int hotelID;
    private String title;
    private String review;
    private String author;
    private int stars;
}
```

Serialization

Deserialization

JSON

```
[
    {
        "hotelID": 1,
        "title": "What a great hotel!",
        "review": "Great hotel,
        "author": "John Smith",
        "stars": 4
    },
    {
        "hotelID": 1,
        "title": "Peaceful night sleep",
        "review": "Would stay again",
        "author": "Kerry Gold",
        "stars": 3
    }
]
```

Making a GET Request using Java

The **RestTemplate** class provides the means with which we can make a request to an API. Here is an example call:

```
RestTemplate restTemplate = new RestTemplate(); // Create a new client
ResponseEntity response = restTemplate.getForEntity(
                "https://api.exchangerate-api.com/v4/latest/USD",
                String.class); // Make GET request using Client
System.out.println(response.getBody()); // your return data returned from .getBody()
```

`response.getBody()` returns a `String` representation of the JSON, just like if you saw the API response in your browser. In the next slide, you'll see how RestTemplate can automatically convert the response data into a Java Object.

Making a GET Request with the Result Mapped to a Java Object

The RestTemplate class provides a way to make a request to an API and treat the result as a Java Object. The JSON return by the API call will be mapped to the specified Java class. Here is an example call:

```
private static final String API_BASE_URL = "http://helpful-site/v1/api/data";
private static RestTemplate restTemplate = new RestTemplate();
MyObj myobj = restTemplate.getForObject(API_BASE_URL, MyObj.class);
```

Note that we can specify the return type of the API call with the second parameter (MyObj.class). Alternatively, if you are getting an array of objects back, we can write the following:

```
MyObj [ ] myobj = restTemplate.getForObject(API_BASE_URL, MyObj[ ].class);
```

Let's Write Some GET Requests and Try Out Some of the Tools We Will be using....