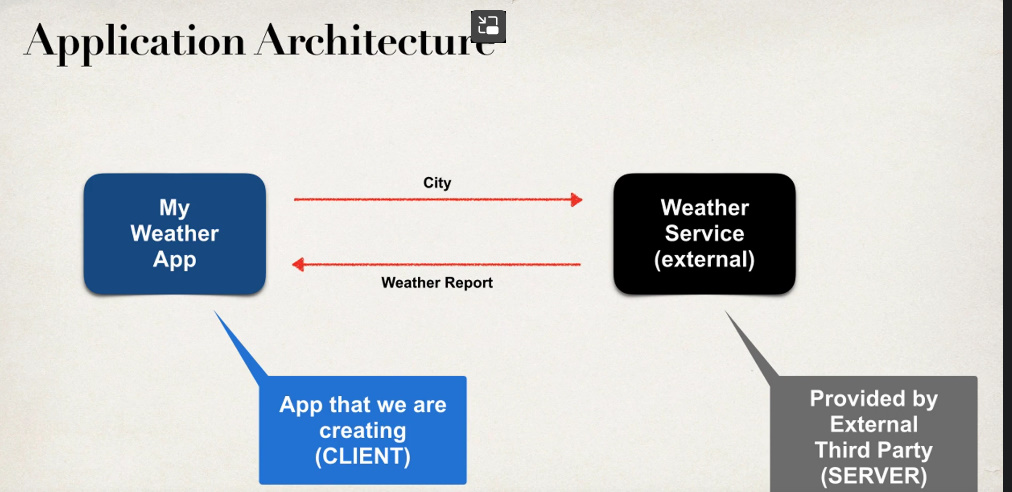
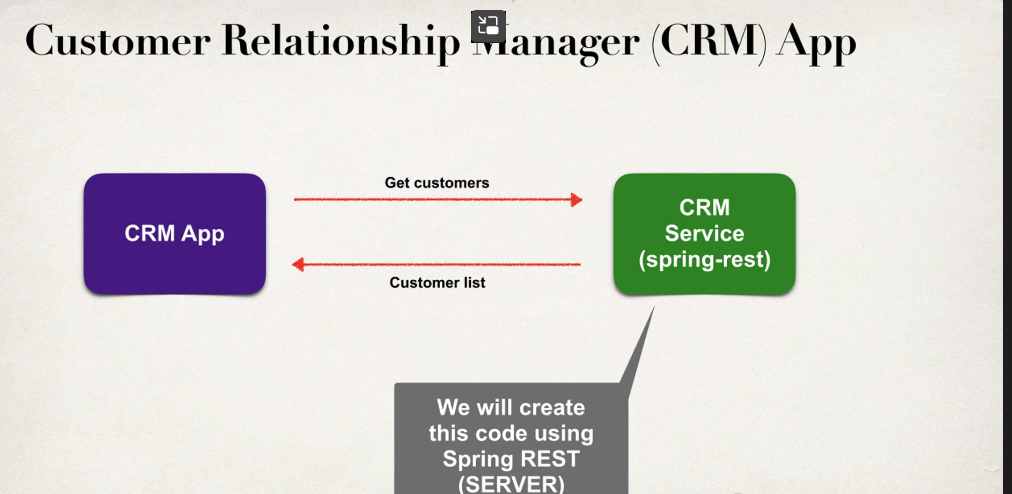
**Ce este REST**

* REST – Representation State Transfer
* REST este o abordare de a comunica intre aplicatii
* Sa zicem ca avem o aplicatie care ne arata prognoza meteo, insa ia aceste date de pe o sursa externa. Aceste date le va putea lua de la sursa externa prin REST API.
* Conexiunea cu sursa se face prin REST API calls prin HTTP
* REST e independent de limbje de programare
* Aplicatiile REST folosesc deobicei XML sau JSON ca data format, dar JSON e mai popular.
* Siteurile daca au suport pentru API, putem sa obtinem orice tip de date dorim de la ele, in diferite formate.

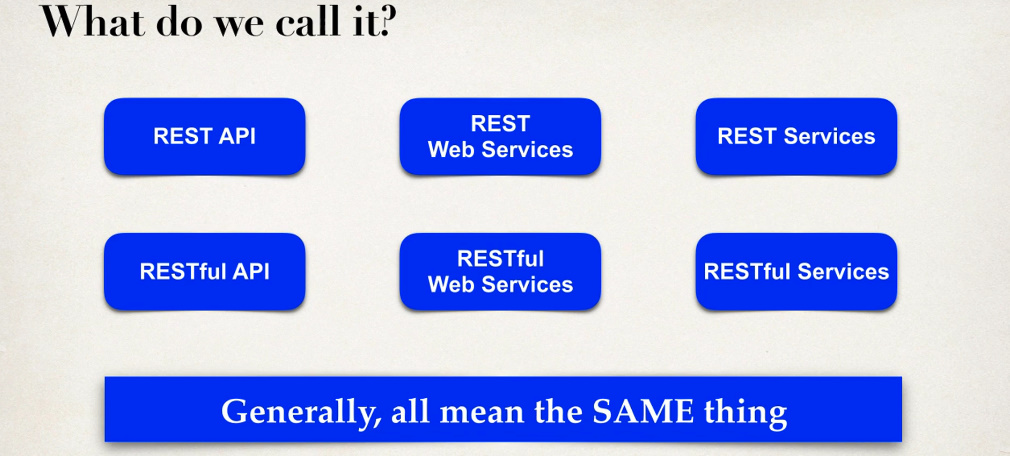


* Alt exemplu ar fi sa avem o aplicatie care converteste diferite valute. Ea va folosi o sursa externa pentru a obtine cursurl valutar si va folosi aceste date.
* Daca de ex am accesa siteul unei retele sociale, am putea folosi REST pentru a primi lista de useri fara a accesa paigina web si sa numaram noi, sub forma de date in JSON.
* Putem sa trimitem noi niste date catre o app si ea sa ne returneze alte date de raspuns, ca de ex sa transformam dolarii in euro din app a noastra folosind o sursa externa

**CRM**

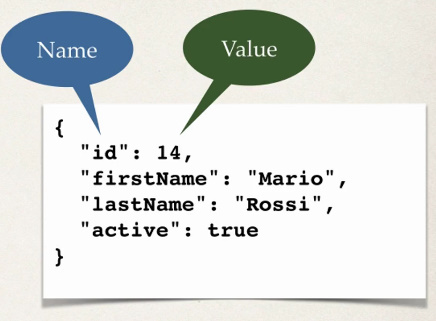
* CRM – Customer Relationship Manager
* 

**REST API vs RESTful API**



**JSON**

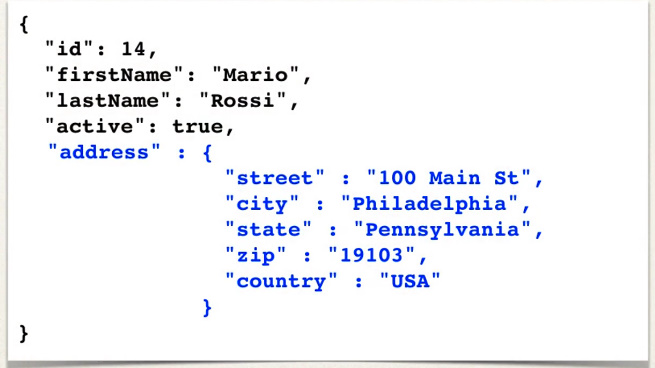
* JSON – javascript object notation
* Este o colectie de name-value pairs.
* Este un data format pentru a face schimb de date
* E independent de orice limbaj, nu doar pentru javascript
* Exemplu:



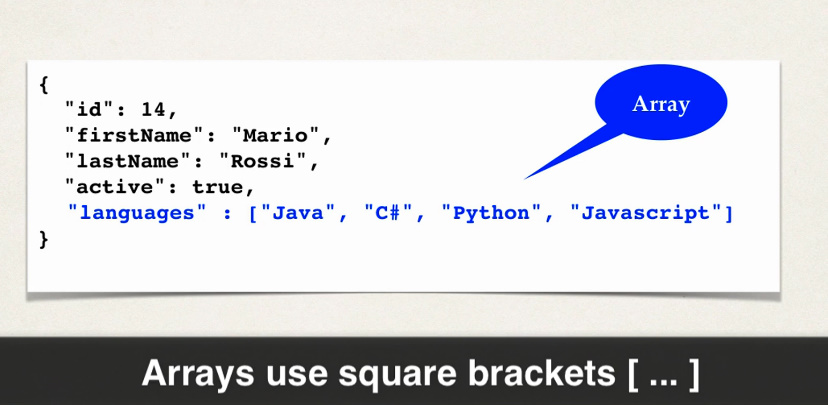
* { } – definesc obiecte
* Obiectele sunt scrise sub forma de pereche

nume : valoare

* Numele membrului e mereu intre „”, nu si valoarea
* Numerele sunt scrise fara „”, daca nu sunt string
* Valorile intre „” sunt mereu stringuri
* Valoarea cu „” e string, fara poate fi int, null, array etc.
* Putem avea si Nested JSON objects, adica mai multe obiecte in unul



* Arrays:

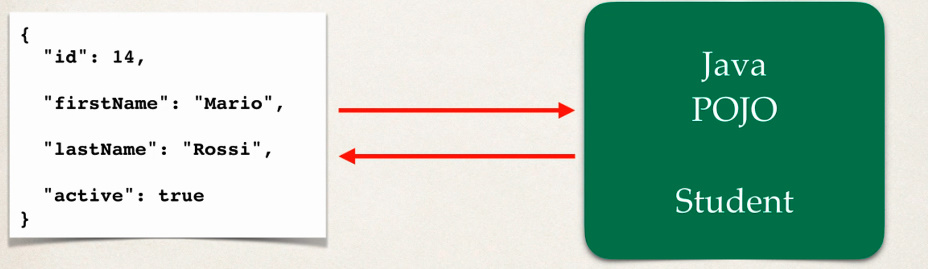


**Java JSON Data Binding**

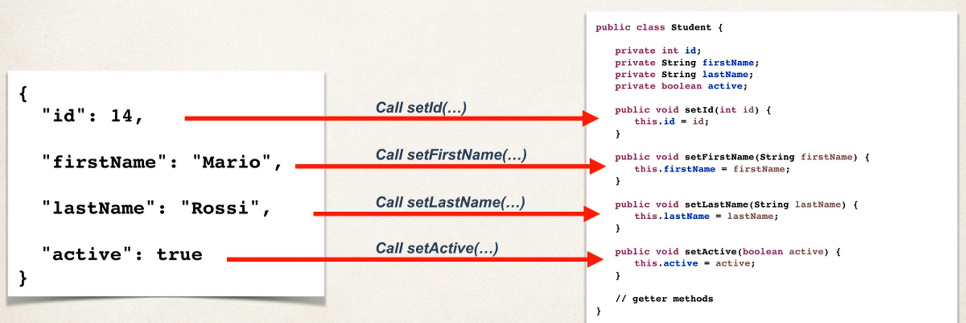
* **Data Binding** – proces de convertire a datelor JSON in JAVA POJO(Plain old Java Object adica obiect simplu) sau invers.
* Data Binding poate fi numit si Mapping, Serialization.
* **Spring** foloseste **Jackson** **Project** in spatele scenei.
* **Jackson** se ocupa de data binding intre JSON si Java POJO

**Jackson DataBinding API**

* Package: com.fasterxml.jackson.databind;
* Jackson suporta si XML
* Jackson este cel care se ocupa de Data Binding
* Jackson face conversia din JSON in POJO si invers, dar le face automat
* Jackson cheama cele mai apropiate getter/setter metode cand face conversia din si in POJO

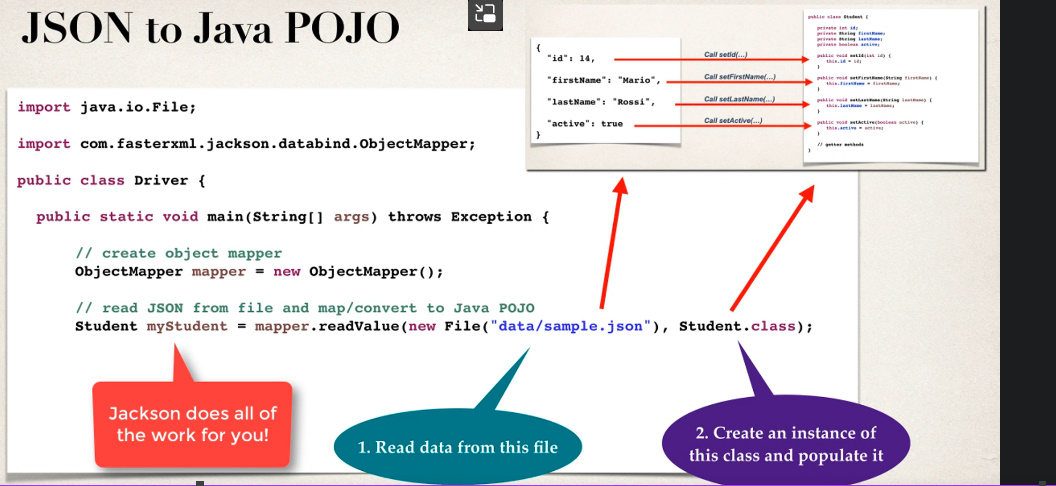


* El va chema metoa setXxx, adica setId, setFirstName. Iarasi atentie ca prima litera e cu majuscula cand se cheama setter/getter!



Jackson niciodata nu apeleaza direct atributele, ci doar prin set si get

**JSON to Java POJO**



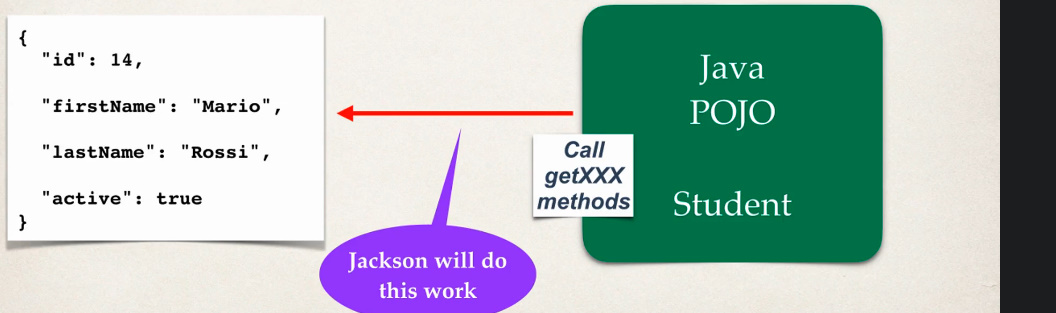
* Avem nevoie de **ObjectMapper** de la Jackson pentru a putea asigura automatizarea procesului de citire,creare si setare.
* MyObj myObj = mapper.readValue(new

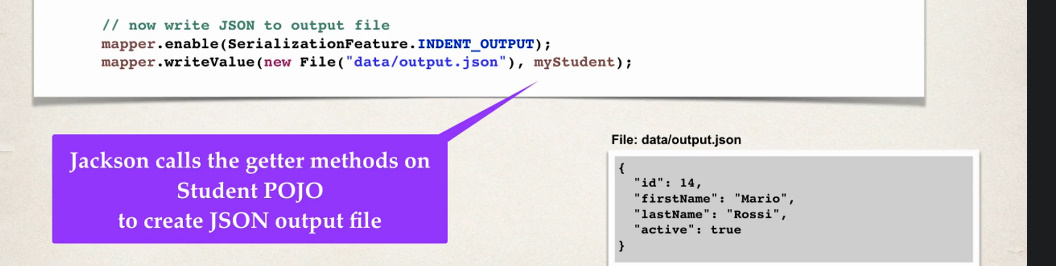
File(location/file.json),MyObj.class);

* Jackson va crea obiectul Student, va citi datele, va accesa setterii si va returna obiectul creat.

**Java POJO to JSON**

* Jackson cheama metodele getter in POJO





* avem nevoie sa-i spunem obiectului ObjectMapper ca acesta urmeaza sa citeasca date .json, deci folosim metoa mapper.enable(SerializationFeature.INDENT\_OUTPUT), dar care e pur si simplu folosita pentru a face datele mai frumos aranjate;
* folosim mapper.writeValue(new File(„locatie/file.json),NumeObject); unde NumeObject e obiectul de la care va lua date.

**Spring and Jacskon Support**

* Cand cream Spring REST apps, Spring se va ocupa automat de Jackson Integration. Adica il va folosi automat.
* JSON data trimise la REST controller sunt convertite in POJO
* Orice java object returnate de REST controller sunt convertite in JSON

**Creare Proiect**

1. Adaugam in pom.xml

<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
 <version>2.14.0</version>  
</dependency>

1. Cream clasa Student
2. public class Student {  
    private int id;  
    private String firstName;  
    private String lastName;  
    private boolean active;  
      
    public Student() {}  
     
    public int getId() {  
    return id;  
    }  
     
    public void setId(int id) {  
    this.id = id;  
    }  
     
    public String getFirstName() {  
    return firstName;  
    }  
     
    public void setFirstName(String firstName) {  
    this.firstName = firstName;  
    }  
     
    public String getLastName() {  
    return lastName;  
    }  
     
    public void setLastName(String lastName) {  
    this.lastName = lastName;  
    }  
     
    public boolean isActive() {  
    return active;  
    }  
     
    public void setActive(boolean active) {  
    this.active = active;  
    }  
   }

3.Cream un folder data, de ex in package la clasa, si cream un fisier .json si adaugam date, ca:

{  
 "id" : 123456,  
 "firstName" : "Eduard",  
 "lastName" : "Mititiuc",  
 "active" :false  
}

1. Cream app:

public class Driver {  
 public static void main(String[] args) {  
 try {  
 ObjectMapper mapper = new ObjectMapper();  
  
 Student student = mapper.readValue(  
 new File("web/WEB-INF/java/data/package.json"), Student.class);  
 System.*out*.println(student.getFirstName());  
 System.*out*.println(student.getLastName());  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

**Nested Objects**

Sa zicem ca avem:

{  
 "id": 14,  
 "firstName": "Mario",  
 "lastName": "Rossi",  
 "active": true,  
 "address": {  
 "street": "100 Main St",  
 "city": "Philadelphia",  
 "state": "Pennsylvania",  
 "zip": "19103",  
 "country": "USA"  
 },  
 "languages" : ["Java", "C#", "Python", "Javascript"]  
}



**Daca vom rula programul de mai sus cu acest .json, vom avea eroare, deoarece Jackson nu va gasi asa metode setter in clasa.**

1. Cream o clasa Address:
2. public class Address {  
    private String street;  
    private String city;  
    private String state;  
    private String zip;  
    private String country;  
     
    public Address(){ }  
     
    public String getStreet() {  
    return street;  
    }  
     
    public void setStreet(String street) {  
    this.street = street;  
    }  
     
    public String getCity() {  
    return city;  
    }  
     
    public void setCity(String city) {  
    this.city = city;  
    }  
     
    public String getState() {  
    return state;  
    }  
     
    public void setState(String state) {  
    this.state = state;  
    }  
     
    public String getZip() {  
    return zip;  
    }  
     
    public void setZip(String zip) {  
    this.zip = zip;  
    }  
     
    public String getCountry() {  
    return country;  
    }  
     
    public void setCountry(String country) {  
    this.country = country;  
    }  
   }
3. Adaugam in Student class:

private Address address;

private String[] languages;

* Daca dorim ca sa fie ignorate campurile necunoscute din JSON, folosim la clasa student anotatia:

**@JsonIgnoreProperties(ignoreUknown=true)**