Accès à une API Rest en java

-- Ressources

Nous avons besoin d'une librairie pour lire du JSON et le convertir en objet Java, et inversement :

- Google GSON

Nous avons également besoin d'effectuer des requêtes (GET, POST, PUT, DELETE...) vers le server Http exposant l'api Rest :

- Apache Http components

-- JSON

-- Modèle

Créer une classe **Model** qui nous servira de classe métier :

- Générer
  - le constructeur par défaut
  - les accesseurs
  - la méthode toString

```
public class Model {
    private int id;
    private String name;
    private boolean access;
    private Date date;
    public Model() {
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public boolean isAccess() {
        return access;
    }
    public void setAccess(boolean access) {
        this.access = access;
    }
    public Date getDate() {
```

```
        return date;
    }
    public void setDate(Date date) {
        this.date = date;
    }

    @Override
    public String toString() {
        return "Model [id=" + id + ", name=" + name + ", access=" + access
                + ", date=" + date + "]";
    }
}
```

-- Classe de test

Créer une classe de test qui va nous permettre de tester GSON :

L'objet **Gson** instancié dans le constructeur nous permettra d'effectuer la conversion dans les 2 sens :

```
public class TestJSON {
    private Gson gson;
    public TestJSON() {
        gson = new GsonBuilder()
        .setDateFormat("yyyy-MM-dd HH:mm:ss")
        .create();
    }
}
```

-- De JSON au model

Créer la méthode suivante retournant une instance de Model construite à partir d'une chaîne JSON :

```
    ...
    public Model jsonToModel(String jsonString){
        return gson.fromJson(jsonString, Model.class);
    }
...
```

Ajouter la méthode Main dans la classe pour la tester :

```
    ...
    public static void main(String args[]){
        TestJSON jsonTest=new TestJSON();
        String jsonStr="{id:1,name:'nom',access:true,date:'2015-03-15 19:22:00'}";
        Model m=jsonTest.jsonToModel(jsonStr);
        System.out.println(m);
    }
...
```

L'exécution doit retourner :

```
Model [id=1, name=nom, access=true, date=Sun Mar 15 19:22:00 CET 2015]
```

-- Du model à JSON

Ajouter la méthode suivante retournant une chaîne JSON construite à partir d'une instance de la classe **Model** :

```
    ...
    public String modelToJson(Model m){
        return gson.toJson(m).toString();
    }
...
```

Ajouter dans la méthode Main le code suivant :

```
    ...
    public static void main(String args[]){
        TestJSON jsonTest=new TestJSON();
        String jsonStr="{id:1,name:'nom',access:true,date:'2015-03-15 19:22:00'}";
        Model m=jsonTest.jsonToModel(jsonStr);
        System.out.println(m);
        m.setName("Autre nom");
        System.out.println(jsonTest.modelToJson(m));
    }
...
```

L'exécution doit retourner :

```
Model [id=1, name=nom, access=true, date=Sun Mar 15 19:22:00 CET 2015]
{"id":1,"name":"Autre nom","access":true,"date":"2015-03-15 19:22:00"}
```

-- Http requests

-- GET

Créer une classe TestHttp :

```
public class TestHttp {
    private Gson gson;
    public TestHttp() {
        gson = new GsonBuilder()
        .setDateFormat("yyyy-MM-dd HH:mm:ss")
        .create();
    }
    public String getHTML(String urlToRead) throws ClientProtocolException,
```

```
IOException {
        String result="";
        CloseableHttpClient httpClient = HttpClients.createDefault();
        try {
            HttpGet getRequest = new HttpGet(urlToRead);
            ResponseHandler<String> responseHandler = new BasicResponseHandler();
            result = httpClient.execute(getRequest, responseHandler);
        }finally {
            httpClient.close();;
        }
        return result;
    }
}
```

-- POST