TD2 Partie2 Projet Maven pour lire des objets au format JSON (librairie jackson)

On se place dans le contexte d'une application de gestion d'un concours. Des fichiers JSON fournis contiennent la description de candidats (nom/numero/score/region). Par exemple, les premières lignes ressemblent à :

```
[
	{
		"nom": "Robert Boone",
		"numero": "1687101477899",
		"score": "82.442799455023",
		"region": 1
	},
	{
		"nom": "Callum Pierce",
		"numero": "1675101251599",
		"score": "70.583646594922",
		"region": 3
	},
	{
		"nom": "Talon Peck",
		"numero": "1637092491999", …
```

On souhaite lire et exploiter ces données dans un projet Maven.

1) **Créer un projet Maven**

   Group ID : pta.sagi
   Artifact ID : maven-json
   Version : 0.1

2) **Ajouter une `classe pta.sagi.concours.entites.Candidat` dont les objets ont les attributs nom/numero/score/region**

```java
package pta.sagi.concours.entites;

public class Candidat
{
	private String nom;
	private long numero;
	private double score;
	private int region;

	public Candidat(){}  // nécessaire

	public Candidat(String nom,long numero,double score,int region){
		this.setNom(nom);
		this.setNumero(numero);
		this.setScore(score);
		this.setRegion(region);
	}
	// getters et setters des attributs nécessaires

	public double getScore() { return score; }

	public void setScore(double score) {   this.score = score; }

	...
```

3) Ajouter la dépendance à la librairie permettant de lire/produire des chaînes JSON

Dans le fichier POM.xml, il doit y avoir une description de la dépendance

```
<dependencies>
<dependency>
	<groupId>com.fasterxml.jackson.core</groupId>
	<artifactId>jackson-databind</artifactId>
	<version>2.9.5</version>
</dependency>
</dependencies>
```

4) Lire le fichier JSON

```
try{
	String fName = "dataJ3.json";  //fichier dans le dossier du projet
	File file = new File(fName);

	ObjectMapper om = new ObjectMapper();
	Candidat[] lesCandidats = om.readValue(file,Candidat[].class);

	...   // exploitation des objets

 }
 catch(Exception e)
 {
	e.printStackTrace();
 }
```

5) Traiter des données de collections à l'aide de l'API Stream (introduction)

La version 8 de Java a introduit les lambdas expressions et une API (Stream) de traitement de collections. On peut appliquer ces fonctionnalités au traitement des candidats lus dans le fichier JSON. Il y a les traitements intermédiaires, qui a un flot (ou séquence) associent un flot, et les traitements finaux qui fournissent (ou non) un résultat et ferment le flot.

Obtenir un objet Stream<T> à partir d'une collection

```
...
Candidat[] lesCandidats = om.readValue(file,Candidat[].class);

Stream<Candidat> sc=Arrays.stream(lesCandidats);
//Stream<Candidat> sc=Stream.of(lesCandidats); // alternative
```

Appliquer un traitement intermédiaire

```
Stream<Candidat> sc=Arrays.stream(lesCandidats);
sc=sc.filter(c->c.getRegion()==2);  // séquence des candidats de région 2
sc=sc.filter(c->c.getScore>70);   // filtre ceux dont le score est >70
```

Appliquer un traitement final

```
sc=Arrays.stream(lesCandidats);
sc = sc.filter(c->c.getNom().contains("Robert"));
sc.forEach(System.out::println);  // traitement final
```

Obtenir une collection/ un tableau

```
Stream<Candidat> sc=Arrays.stream(lesCandidats);
sc=sc.filter(c->c.getRegion()==1); //filtre les candidats

List<Candidat> lc=sc.collect(Collectors.toList()); // obtient une collection

// Object[] tabC = sc.toArray();  //ou obtient un tableau Object[]
```

Exemples à programmer

a) Afficher les candidats de la région 3 dont le score est entre 70 et 80

b) Obtenir une collection contenant les noms de ces candidats

c) Obtenir une collection contenant tous les scores de la région 4

d) Obtenir le candidat qui a le meilleur score