Le but de ce tp est de questionner une API Rest pour obtenir des horaires de train.

Le site de la SNCF fournit une API Rest à l'adresse `https://api.sncf.com/v1/coverage/sncf/`, documenté à l'adresse `https://data.sncf.com/api`.

1 Établir la connexion

Question 1 Créé une instance d'URL avec la valeur "https ://api.sncf.com/v1/coverage/sncf". Créé une instance de HttpsURLConnection avec URL.openConnection(). Imprimer le résultat de la méthode getResponseCode() qu'en concluez-vous ?

Cette API demande un token de sécurité en authentification basic pour ce connecter, nous utiliserons "82ee4aa2-ecf2-4844-b9a1-0bcbff7a285e" comme nom d'utilisateur et le mot de passe vide.

L'authentification basique consiste à ajouter un champ "Authorization" dans l'en-tête de connexion ayant pour valeur "Basic +mdp" ou mdp est un encodage en base64 de la chaine de caractère constitué du nom d'utilisateur puis " :" puis le mot de passe.

Question 2 Ajouter l'authentification à la méthode précédente.
On utilisera Base64.getEncoder().encodeToString(**byte**[] mdp) pour fabriquer l'encodage de la chaine d'authentification.
On utilisera HttpUrlConnection.setRequestProperty(String key, String value) pour ajouter un champ à l'en tête HTTP. Vérifier que vous obtenez "200" comme code de retour pour la connexion.

Dans la suite, nous allons utiliser le parseur de json "Gson" que vous pouvez télécharger ici : `http://search.maven.org/#artifactdetails%7Ccom.google.code.gson%7Cgson%7C2.6.2%7Cjar`. Vous pouvez aussi utilser la librairie du dernier TP si vous êtes sure de votre implémentation.

Question 3 Récupérer le contenu de la page web comme un élément Json avec :

```
InputStream is = connection.getInputStream();
InputStreamReader isr = new InputStreamReader(is);
JsonParser parser = new JsonParser();
JsonElement element = parser.parse( isr);
```

Afficher le contenu de element.

Question 4 Écrire une méthode de signature :
public static JsonElement get(String s) **throws** IOException qui contient tout le code précédemment écris telle que System.out.println(get("places")); imprime le contenu de `https://api.sncf.com/v1/coverage/sncf/places"`. Faites attention à la gestion des erreurs en testant le code de retour HTTP, vous aurez besoin de HttpUrlConnection.getErrorStream().

2 Recherche

Pour faire une recherche dans cette API on utilise l'adresse : `https://api.sncf.com/v1/coverage/sncf/places?q={recherche}"` ou "{recherche}" et le texte recherché encodé comme une URL.

Question 5 Faites une recherche sur “Créteil” et affichez le résultat

Dans l’api une gare est une “place” dont le champ “embedded_type” est égal à “stop_area”.

Question 6 Écrire une fonction getGares(String s) qui fait une recherche sur la chaine de caractère s et qui renvoie un Map<String,String> Contenant le champ “id” comme clé et le champ “name” comme valeur. *Pour encoder une URL url on peut utiliser la méthode statique* URLEncoder.encode(url, "UTF−8").

Pour afficher les prochains départs dans cette API on utilise : `https://api.sncf.com/v1/coverage/sncf/stop_areas/{id_gare}/departures`

Question 7 Regarder un example de réponse du serveur sur la liste des départs.

Question 8 Écrire une classe Departure avec un champ String name et un champ Date time. Écrire un constructeur à partir d’un JsonElement e , tel que le nom soit initialisé à la valeur contenue dans e.route.name et l’horaire au contenue de e.stop_date_time.departure_date_time. Définir une méthode toString()
Vous pouvez utiliser **new** SimpleDateFormat("yyyyMMdd’T’HHmmss") *pour parser la date.*

Question 9 Afficher les prochains départs depuis la gare “Le vert de maison”.