Nous allons réaliser une série d'exercices qui nous permettra d'étudier et de manipuler divers concepts de l'environnement d'Apache Kafka.

Contexte du projet: Une application reçoit des messages sur l'achat de médicaments. Un tel message contient des informations sur l'acheteur (nom et prénom), la référence (un code nommé CIP) et le prix du produit acheté (calculé depuis un prix de base + ou – 10%) et le lieu de l'achat (une pharmacie). A la réception de ces messages, vous devez effectuer plusieurs traitements sur ces derniers. Les traitements correspondent à la somme des ventes, l'anonymisation des données de la transaction (message), différentes analyses des transactions (par exemple, la volumétrie des achats par pharmacie, région).

Vous allez créer des clients factices avec la librairie JavaFaker.
Les données sur les médicaments et les pharmacies proviendront d'une source de données correspondant à un système de gestion de base de données relationnel (SGBR, le dump SQL provient de PostgreSQL). La base contient les deux tables suivantes:

```
            Table "public.drugs4projet"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 cip    | integer |           | not null |
 prix   | real    |           |          |
Indexes:
    "drugs4projet_pkey" PRIMARY KEY, btree (cip)

                Table "public.pharm4projet"
 Column  |      Type      | Collation | Nullable | Default
---------+----------------+-----------+----------+---------
 id      | integer        |           | not null |
 nom     | text           |           |          |
 adresse | text           |           |          |
 depart  | character(100) |           |          |
 region  | character(100) |           |          |
Indexes:
    "pharm4projet_pkey" PRIMARY KEY, btree (id)
```

La base de données contient 136 médicaments et 588 pharmacies.
La séquence de production de messages est la suivante:
- création avec JavaFaker du nom et prénom du client
- sélection aléatoire d'un code CIP depuis la table drugs4Projet et adaptation aléatoire du prix du médicament (+ ou – 10% du prix stocké dans la base de données)
- sélection aléatoire d'une pharmacie.

L'implantation se fera en Java et les dépendances nécessaires sont les suivantes (extrait du pom.xml du projet):

```
<properties>
  <maven.compiler.source>11</maven.compiler.source>
  <maven.compiler.target>11</maven.compiler.target>
  <kafka.version>3.1.0</kafka.version>
```

```
</properties>
<dependencies>
   <!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients -->
   <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-clients</artifactId>
      <version>${kafka.version}</version>
   </dependency>
   <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-streams</artifactId>
      <version>${kafka.version}</version>
   </dependency>
   <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.10.0</version>
   </dependency>
   <dependency>
      <groupId>com.github.javafaker</groupId>
      <artifactId>javafaker</artifactId>
      <version>0.16</version>
   </dependency>
   <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <version>42.2.8</version>
   </dependency>
   <dependency>
      <groupId>com.twitter</groupId>
      <artifactId>bijection-avro_2.13</artifactId>
      <version>0.9.7</version>
   </dependency>
</dependencies>
```

Si vous ne disposez pas de Postgresql, vous pouvez utiliser un autre SGBDR (eg mySQL).

1. Dans un premier temps, vous devez produire les messages au format JSON avec l'API Producer de Kafka. Vous utiliserez le processeur JSON Jackson. Voici un exemple de message:
`{"nom":"Rodriguez","prenom":"Ottis","cip":3846200,"prix":2.56,"idPharma":151}`
Vous devez ensuite consommer les enregistrements depuis le topic alimenté par le Producer et affichez les informations à l'écran

2. A la suite du cours sur la mise en oeuvre, et en particulier de la partie sur le format Avro, vous devez adapter les codes du producer et du consumer de la question 1 pour gérer les enregistrements au format Avro.
Il vous sera nécessaire de créer un schéma Avro (extension avsc).
L'approche classique correspond à définir la sérialisation de la valeur d'un message avec AvroSerialization. Néanmoins, cette approche implique de charger la plateforme Confluent (plus de 900Mo). Vous utilisez une autre approche, plus légère, basée sur la bibliothèque bijection proposée par Twitter (https://github.com/twitter/bijection).
Pour le moment, on se limite à la bonne réception des messages produits.

3. Vous changez la configuration de Kafka. On veut maintenant avoir 3 brokers, 3 partitions et un replication factor de 3.

3.1 Modifier le producer pour qu'il produise sur les 3 partitions d'un nouveau topic

3.2 Modifier le consommateur précédent pour qu'il consomme depuis les 3 partitions, affiche les transactions d'achat de médicaments et produise dans un nouveau topic (on le nomme top2 dans la suite de l'énoncé mais vous pouvez le nommer comme vous voulez) dans l'objectif de faire l'analyse des ventes par médicament/

3.3 Vous devez créer un groupe de consommateur de 3 consommateurs qui consommera le messages de top2 dans l'objectif de compter le cumul des ventes par médicament. Vous afficherez le cumul du médicaments qui a été impliqués dans la transaction "streamée"

A la suite de la partie du cours sur Kafka Streams, vous implanterez les questions suivantes, avec une sérialization des flux en JSON.

5. Anonymisation des ventes
Vous allez concevoir une topologie de processeurs Kafka Streams. Un premier processeur sera utilisez par de nombreux autres processors de la topologie. Ce processor générera un nouveau topic contenant les transactions anonymisés, c'est-à-dire que le nom et le prénom de chaque message seront remplacés par des « *** ».

6. Sélection de médicaments
A partir du topic anonymisé, vous devez concevoir un nouveau consumer Kafka qui n'affichera que les transactions dont le prix du médicament sera supérieur à 4 euros.

7. Vous devez ajouter la valeur de la région de chaque pharmacie dans le flux à l'aide d'une jointure.

```java
props.put(ConsumerConfig.GROUP_ID_CONFIG,
        "KafkaStreamExampleConsumer");
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
        LongDeserializer.class.getName());
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
        StringDeserializer.class.getName());

StreamsConfig streamsConfig = new StreamsConfig(props);
Serde<String> stringSerde = Serdes.String();
Serde<RecordGenerator> recordSerde = Serdes.serdeFrom(RecordGenerator.class);
StreamsBuilder builder = new StreamsBuilder();

KStream<String, RecordGenerator> kStream = builder.stream(KConfig.TOPIC, Consumed.with(stringSerde, recordSerde));

KStream<String, RecordGenerator> anonymousKStream = kStream.map((k, v) -> {
    v.setNom("**");
    v.setPrenom("**");
    return KeyValue.pair(k, v);
});

anonymousKStream.to( s "test-out", Produced.with(stringSerde, recordSerde));

final KafkaStreams kafkaStreams = new KafkaStreams(builder.build(), streamsConfig);
kafkaStreams.start();
kafkaStreams.close();
```