Table of Contents

1. **First Stream Demo** 1
2. **UserStream** 2
3. **ASSIGNMENT 1:** 5
4. **ASSIGNMENT 2:** 5
5. **REFERENCES** 5

1. First Stream Demo

1.1. Create FirstKafkaStream Maven quickstart project.
1.2. Add below dependencies:

```
<properties>
            <kafka.version>3.5.1</kafka.version>
</properties>

<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
    <version>${kafka.version}</version>
  </dependency>
<dependency>
    <groupId>commons-lang</groupId>
    <artifactId>commons-lang</artifactId>
    <version>2.6</version>
  </dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.14.2</version>
</dependency>
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>${kafka.version}</version>
</dependency>
```

1.3. Add win-scripts as usual.

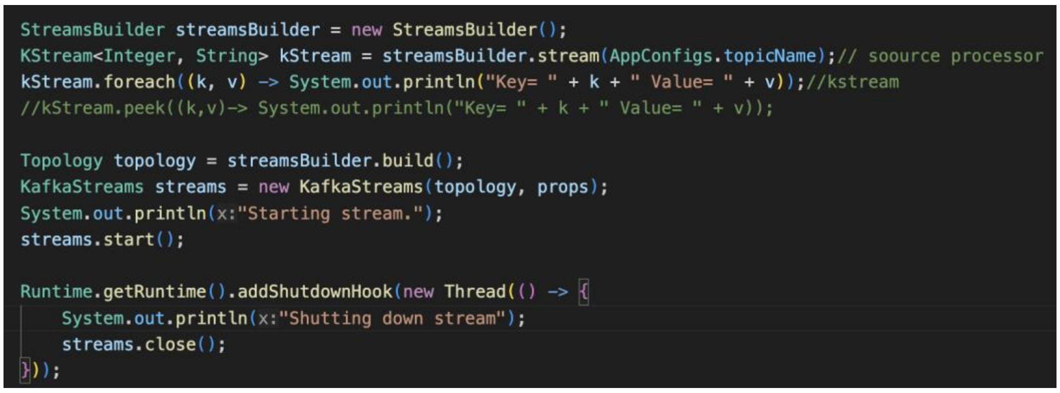
1.4. Create class AppConfigs as follows:

```
public class AppConfigs {
   public final static String applicationID = "HelloStream";
   public final static String bootstrapServers = "localhost:9092,localhost:9093";

   public  final  static String topicname = "invoice";

 }
```

1.5. Create a HelloStreams class as follows:



```
StreamsBuilder streamsBuilder = new StreamsBuilder();
KStream<Integer, String> kStream = streamsBuilder.stream(AppConfigs.topicName);// soource processor
kStream.foreach((k, v) -> System.out.println("Key= " + k + " Value= " + v));//kstream
//kStream.peek((k,v)-> System.out.println("Key= " + k + " Value= " + v));

Topology topology = streamsBuilder.build();
KafkaStreams streams = new KafkaStreams(topology, props);
System.out.println(x:"Starting stream.");
streams.start();

Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    System.out.println(x:"Shutting down stream");
    streams.close();
}));
```

1.6. Do run Producer created in earlier session to produce data for HelloStream to consume

2. UserStream

Add below samples in FirstKafkaStream project created earlier

2.1. Copy user.json schema from JsonToAvroProject and add the jsonschema2pojo plugin in pom.xml file
2.2. Run the maven compile phase to create the respective POJO

NOTE: [Make sure the javaType path is same as your project package structure]

2.3. Create 2 topics valid-user and invalid-user and 2 CLI consumers to read the valid and invalid users within the win-scripts folder

2.4. Create Json serializer as follows:

```
public class JsonSerializer<T> implements Serializer<T> {

 private final ObjectMapper objectMapper = new ObjectMapper();

  public JsonSerializer() {

  }

  @Override
  public void configure(Map<String, ?> config, boolean isKey) {
     //Nothing to Configure
  }

  /**
   * Serialize JsonNode
   *
   * @param topic Kafka topic name
   * @param data  data as JsonNode
```

```
     * @return byte[]
     */
    @Override
    public byte[] serialize(String topic, T data) {
        if (data == null) {
            return null;
        }
        try {
            return objectMapper.writeValueAsBytes(data);
        } catch (Exception e) {
            throw new SerializationException("Error serializing JSON message", e);
        }
    }

    @Override
    public void close() {

    }
}
```

2.5. Create Json Deserializer as follows:

```
public class JsonDeserializer<T> implements Deserializer<T> {

    private ObjectMapper objectMapper = new ObjectMapper();
private Class<T> className;
public static final String KEY_CLASS_NAME_CONFIG = "key.class.name";
public static final String VALUE_CLASS_NAME_CONFIG = "value.class.name";

public JsonDeserializer() {    }

/**
 * Set the specific Java Object Class Name
 * @param props set specific.class.name to your specific Java Class Name
 * @param isKey set it to false
 */
@SuppressWarnings("unchecked")
@Override
public void configure(Map<String, ?> props, boolean isKey) {
    if (isKey)
        className = (Class<T>) props.get(KEY_CLASS_NAME_CONFIG);
    else
        className = (Class<T>) props.get(VALUE_CLASS_NAME_CONFIG);
}

/**
 * Deserialize to a POJO
 * @param topic topic name
 * @param data  message bytes
 * @return Specific Java Object
 */
@Override
public T deserialize(String topic, byte[] data) {
    if (data == null) {
        return null;
    }
    try {
        return objectMapper.readValue(data, className);
    } catch (Exception e) {
        throw new SerializationException(e);
      }
    }
    @Override
    public void close() {
      //nothing to close
    }
}
```

2.6. Create AppSerde as follows:

```
public class AppSerdes extends Serdes {

  static final class UserSerde extends WrapperSerde<User> {
    UserSerde() {
      super(new JsonSerializer<>(), new JsonDeserializer<>());
    }
  }
  public static Serde<User> User() {
    UserSerde serde = new UserSerde();
    Map<String, Object> serdeConfigs = new HashMap<>();
    serdeConfigs.put(JsonDeserializer.VALUE_CLASS_NAME_CONFIG, User.class);
    serde.configure(serdeConfigs, false);

    return serde;
  }
}
```

2.7. Create a class UserStream and add the below code for kafka streams to process valid and invalid users

```
Properties props = new Properties();
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "User STREAM");
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, AppConfigs.bootstrapServers);

StreamsBuilder streamsBuilder = new StreamsBuilder();

KStream<Integer, User> stream = streamsBuilder.stream("user-topic",
        Consumed.with(Serdes.Integer(), AppSerdes.User()));

stream.filter((k,v)-> v.getAge() >=20)
        .peek((k,v)-> System.out.println("valid "+k+" "+v.getAge()))
        .to("valid-user-topic", Produced.with(Serdes.Integer(), AppSerdes.User()));

stream.filter((k,v)-> v.getAge() < 20)
        .peek((k,v)-> System.out.println("Invalid "+k+" "+v.getAge()))
        .to("invalid-user-topic", Produced.with(Serdes.Integer(), AppSerdes.User()));

Topology topology = streamsBuilder.build();
KafkaStreams streams = new KafkaStreams(topology, props);
System.out.println("Starting stream.");
try {
    streams.start();
}catch (Exception e)
{
    System.out.println("error "+e.getMessage());
}
Thread.sleep(3000);
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    System.out.println("Shutting down stream");
    streams.close();
}));
```

2.8. Run the JsonAvroToPOJO project and produce some user data
2.9. Then run UserStreams and check the valid and invalid topic for the filtered users.

3. ASSIGNMENT 1:

3.1. Preparatory phase
- 3.1.1. Go through the JsonInvoiceProject shared.
- 3.1.2. Change the mac-scripts to win-scripts.
- 3.1.3. Run zookeeper, 3 brokers, create command for the topic pos.
- 3.1.4. Run the PosSimulator class to produce invoices. This runs in an infinite loop hence you can terminate after some time to stop producer from generating invoices.

3.2. TASK TO DO:
- 3.2.1. Take a look at PosValidator class. It filters the valid amd invalid records
- 3.2.2. Convert this class to use streams.
- 3.2.3. Add respective streams related dependencies in this project to create kafka streams for PosValidator.
- 3.2.4. Do not forget to add json serializer, deserializer and Serde for PosInvoice class.

4. ASSIGNMENT 2:

4.1. Open the JsonInvoiceKafkaStreamProject and complete the TODOS within the PosFanoutApp.java class. The business requirement and the corresponding business logic is already implemented.
4.2. Create streams based on the requirements

5. REFERENCES

- https://medium.com/@agvillamizar/implementing-custom-serdes-for-java-objects-using-json-serializer-and-deserializer-in-kafka-streams-d794b66e7c03
- https://nuwancs.medium.com/kafka-kstream-joins-for-json-objects-39ad2c31a51c