Session 23: More Kafka Assignment 1

Task 1:

Create a java program MyKafkaProducer.java that takes a file name and delimiter as input arguments.
It should read the content of file line by line.
Fields in the file are in following order
1. Kafka Topic Name
2. Key
3. value
For every line, insert the key and value to the respective Kafka broker in a fire and forget mode.
After record is sent, it should print appropriate message on screen.
Pass dataset_producer.txt as the input file and -as delimiter.
LINK: https://drive.google.com/file/d/0B_Qjau8wv1KoSnR5eHpKOF9rTFU/view?usp=sharing

```
import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Properties;
import java.util.concurrent.ExecutionException;

import org.apache.kafka.clients.producer.RecordMetadata;

public class KafkaFileProducer extends Thread {

   private static final String topicName
         = "test";
   public static final String fileName = "dataset_producer.txt";

   private final KafkaProducer<String, String> producer;
   private final Boolean isAsync;

   public KafkaFileProducer(String topic, Boolean isAsync) {
      Properties props = new Properties();
```

```
    props.put("bootstrap.servers", "localhost:9092");
    props.put("client.id", "DemoProducer");
    props.put("key.serializer",
            "org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer",
            "org.apache.kafka.common.serialization.StringSerializer");
    producer = new KafkaProducer<String, String>(props);
    this.isAsync = isAsync;
}

public void sendMessage(String key, String value) {
    long startTime = System.currentTimeMillis();
    if (isAsync) { // Send asynchronously
        producer.send(
                new ProducerRecord<String, String>(topicName, key),
                (Callback) new DemoCallBack(startTime, key, value));
    } else { // Send synchronously
        try {
            producer.send(
                    new ProducerRecord<String, String>(topicName, key, value))
                    .get();
            System.out.println("Sent message: (" + key + ", " + value + ")");
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
    }
}

public static void main(String [] args){
    KafkaFileProducer producer = new KafkaFileProducer(topicName, false);
    int lineCount = 0;
    FileInputStream fis;
    BufferedReader br = null;
    try {
        fis = new FileInputStream(fileName);
        //Construct BufferedReader from InputStreamReader
        br = new BufferedReader(new InputStreamReader(fis));

        String line = null;
        while ((line = br.readLine()) != null) {
            lineCount++;
            producer.sendMessage(lineCount+"", line);
        }

    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }finally{
        try {
```

```
            br.close();
          } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
          }
        }

    }
}

class DemoCallBack implements Callback {

    private long startTime;
    private String key;
    private String message;

    public DemoCallBack(long startTime, String key, String message) {
        this.startTime = startTime;
        this.key = key;
        this.message = message;
    }

    /**
     * A callback method the user can implement to provide asynchronous handling
     * of request completion. This method will be called when the record sent to
     * the server has been acknowledged. Exactly one of the arguments will be
     * non-null.
     *
     * @param metadata
     *          The metadata for the record that was sent (i.e. the partition
     *          and offset). Null if an error occurred.
     * @param exception
     *          The exception thrown during processing of this record. Null if
     *          no error occurred.
     */
    public void onCompletion(RecordMetadata metadata, Exception exception) {
        long elapsedTime = System.currentTimeMillis() - startTime;
        if (metadata != null) {
            System.out.println("message(" + key + ", " + message
                    + ") sent to partition(" + metadata.partition() + "), "
                    + "offset(" + metadata.offset() + ") in " + elapsedTime
                    + " ms");
        } else {
            exception.printStackTrace();
        }
    }
}
```

Task 2:

Modify the previous program MyKafkaProducer.java and create a new Java program KafkaProducerWithAck.java
This should perform the same task as of KafkaProducer.java with some modification.
When passing any data to a topic, it should wait for acknowledgement.
After acknowledgement is received from the broker, it should print the key and value which has been
written to a specified topic.
The application should attempt for 3 retries before giving any exception.
Pass dataset_producer.txt as the input file and -as delimiter.

```
public class KafkaConsumerExample {
  ...
    static void runConsumer() throws InterruptedException {
        final Consumer<Long, String> consumer = createConsumer();
        final int giveUp = 100;   int noRecordsCount = 0;
        while (true) {
            final ConsumerRecords<Long, String> consumerRecords =
                    consumer.poll(1000);
            if (consumerRecords.count()==0) {
                noRecordsCount++;
                if (noRecordsCount > giveUp) break;
                else continue;
            }
            consumerRecords.forEach(record -> {
                System.out.printf("Consumer Record:(%d, %s, %d, %d)\n",
                        record.key(), record.value(),
                        record.partition(), record.offset());
            });
            consumer.commitAsync();
        }
        consumer.close();
        System.out.println("DONE");
    }
}
```

```
package articlestreamer.kafka

import org.apache.kafka.clients.producer.{Callback, RecordMetadata}

class RecordCallback extends Callback {

  override def onCompletion(metadata: RecordMetadata, ex: Exception) = {
    if (ex != null) {
      handleException(ex)
    } else {
      println(s"Successfully sent message : $metadata")
    }
```

```
  }

  def handleException(exception: Exception): Unit = {
    Console.err.println(s"Error while attempting to send message : $exception")
  }
}
```

```
package articlestreamer.kafka

import org.apache.kafka.clients.producer.{Callback, RecordMetadata}

class RecordCallback extends Callback {

  override def onCompletion(metadata: RecordMetadata, ex: Exception) = {
    if (ex != null) {
      handleException(ex)
    } else {
      println(s"Successfully sent message : $metadata")
    }
  }

  def handleException(exception: Exception): Unit = {
    Console.err.println(s"Error while attempting to send message : $exception")
  }
}
```

Error while downloading dataset, hence considered a different dataset..