Kafka Java Programming.

Create a new maven project (make sure jdk id 1.8)

2. Inside the pom.xml file add these two dependencies. We are using kafka-cleints 2.0.x and slf4j simple 1.7.25

```
<!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients -->
   <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-clients</artifactId>
      <version>2.0.0</version>
   </dependency>
   <!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-simple -->
   <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-simple</artifactId>
      <version>1.7.25</version>
      <!-- <scope>test</scope> -->
   </dependency>
```

3. Comment the <scope>test</scope> in the dependency for slf4j
4. Create a package inside src/main/java (com.github.Harshali15.tutorial1)
5. Create a java file under this package (App.java)
6. Write a simple hello world code. It should run successfully.

*****Creating Java producer****

There are 3 steps in creating a producer:
1. Create producer properties
2. Create producer
3. Send data
4. flush data

1. Create producer properties

```
String bootstrap_server = "localhost:9092";

Properties properties = new Properties();
properties.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrap_server);
properties.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
properties.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
```

2. Create producer

```
KafkaProducer<String,String> producer = new KafkaProducer<>(properties);

//create a producer record
ProducerRecord<String,String> record = new ProducerRecord<String,String>("topic1","trying java kafka 2");
```

3. send data --asynchronous- means it is happening in the background so you have to wait, flush()

```
producer.send(record);
```

4. flush data

```
producer.flush();
```

5. flush and close producer
producer.close();

5. Now connect to your kafka zookeper and server and open a consumer which listens to topic 1. When you run App.java now you will see the message appearing in the consumer

*****Producer with callback****
To understand where the message was produced, partiton no, timestamp, offset etc.
In the producer.send function add a callback and print out reqd info
Note there is no ordering of messages received in consumer because we are not passing any key
messages are being produced to 2 partitons and are ordered within partition

****Producer with keys****
Add topic,key,value instead of topic,value in ProducerRecord
ProducerRecord<String,String> record = new ProducerRecord<String,String>(topic,key,value);
Use .get() with send to make the send synchronous. **Not to be used in Production at all ***

// data with specific keys will always go to same partiton now
// eg: id 0,1,2,3,5,7,8 -->partition 1
// eg: id 4,6,9 -->partition 0

***Creating Java Consumer**

1. create consumer configs/properties
Properties properties = new Properties();
properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrap_server);
properties.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
properties.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
properties.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
properties.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest"); //earliest/latest/none

2. create consumer
KafkaConsumer<String,String> consumer = new KafkaConsumer<String,String>(properties);

3. subscribe consumer to topic
//consumer.subscribe(Collections.singleton("topic1")); //to subscribe to single topic
//consumer.subscribe(Arrays.asList("topic1","topic2")); //subscribe to mutiple topics
consumer.subscribe(Arrays.asList(topic));

4. poll for data
  consumer does not get data until it asks for data

```
while(true){
      ConsumerRecords<String,String> records = consumer.poll(Duration.ofMillis(100));

      for(ConsumerRecord<String,String> record : records) {
         System.out.println("Key" + record.key() +
         "\n" + "Value : "+ record.value() +
         "\n"+ "Offset : " + record.offset() +
         "\n"+ "Partiton : "+ record.partition()
         );
      }
  }
```

note : consumer will read partition wise means read all data from each partiton at once before moving to next partition

when new data comes it is read in the order it arrives

Consumer groups

By changing group id you basically reset the application

To run multiple consumers start multiple instances of the consumergroup.java. Start prodcuergroup.java. You will notice that each consumer will read specific partitions only. If one oro other consumer goes down then it is rebalanced amongst other consumers.

Java consumer with threads

1. Go through he ConsumerWithThreads.java code / video for the same

Assign seek
1. assign and seek are used to replay data or fetch a speicifc message

```
//assign
TopicPartition partitionToReadFrom = new TopicPartition(topic, 0);
long offsetToReadFrom = 15L;
consumer.assign(Arrays.asList(partitionToReadFrom));

//seek
consumer.seek(partitionToReadFrom,offsetToReadFrom);

int numMessagesToRead = 6;
boolean keepReading = true;
int numOfMessagesReadAlready = 0;
while(keepReading){
    ConsumerRecords<String,String> records = consumer.poll(Duration.ofMillis(100));
    for(ConsumerRecord<String,String> record : records) {
        numOfMessagesReadAlready++;
        System.out.println("Key" + record.key() +
        "\n" + "Value : "+ record.value() +
        "\n"+ "Offset : " + record.offset() +
        "\n"+ "Partiton : "+ record.partition()
        );
        if(numOfMessagesReadAlready>numMessagesToRead){
            keepReading=false;
            break;
        }
    }
}
```