1. Configurer le stream

```
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.StreamsConfig;

import java.util.Properties;

public class StreamConfiguration {

    public static Properties getConfiguration() {
        Properties properties = new Properties();
        properties.put(StreamsConfig.APPLICATION_ID_CONFIG, "bank-balance");
        properties.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");
        properties.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
        properties.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
        properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        properties.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, "0");
        return properties;
    }
}
```

2. Model

a. JsonSerde

```
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.SneakyThrows;
import org.apache.kafka.common.serialization.Deserializer;
import org.apache.kafka.common.serialization.Serde;
import org.apache.kafka.common.serialization.Serializer;

public class JsonSerde<T> implements Serde<T> {

    public static final ObjectMapper OBJECT_MAPPER = new ObjectMapper();
    private final Class<T> type;

    public JsonSerde(Class<T> type) {
        this.type = type;
    }

    @Override
    public Serializer<T> serializer() {
        return (topic, data) -> serialize(data);
    }

    @SneakyThrows
    private byte[] serialize(T data) {
```

```
        return OBJECT_MAPPER.writeValueAsBytes(data);
    }

    @Override
    public Deserializer<T> deserializer() {
        return (topic, bytes) -> deserialize(bytes);
    }

    @SneakyThrows
    private T deserialize(byte[] bytes) {
        return OBJECT_MAPPER.readValue(bytes, type);
    }
}
```

b. BankTransaction

```
import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.math.BigDecimal;
import java.util.Date;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class BankTransaction {

    private Long id;
    private Long balanceId;
    private BigDecimal amount;
    @JsonFormat(shape = JsonFormat.Shape.STRING,
            pattern = "dd-MM-yyyy hh:mm:ss")
    public Date time;
    @Builder.Default
    public BankTransactionState state = BankTransactionState.CREATED;

    public static enum BankTransactionState {
        CREATED, APPROVED, REJECTED
    }
}
```

c. BankBalance

```
import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.AllArgsConstructor;
```

```
import lombok.Data;
import lombok.NoArgsConstructor;

import java.math.BigDecimal;
import java.util.Date;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class BankBalance {

   private Long id;
   private BigDecimal amount = BigDecimal.ZERO;
   @JsonFormat(shape = JsonFormat.Shape.STRING,
         pattern = "dd-MM-yyyy hh:mm:ss")
   private Date lastUpdate;
   private BankTransaction latestTransaction;

   public BankBalance process(BankTransaction bankTransaction) {
      this.id = bankTransaction.getBalanceId();
      this.latestTransaction = bankTransaction;
      if(this.amount.add(bankTransaction.getAmount()).compareTo(BigDecimal.ZERO) >= 0) {
         this.latestTransaction.setState(BankTransaction.BankTransactionState.APPROVED);
         this.amount = this.amount.add(bankTransaction.getAmount());
      } else {
         this.latestTransaction.setState(BankTransaction.BankTransactionState.REJECTED);
      }
      this.lastUpdate = bankTransaction.getTime();
      return this;
   }
}
```

3. BankBalanceTopology

```
import com.github.programmingwithmati.model.BankBalance;
import com.github.programmingwithmati.model.BankTransaction;
import com.github.programmingwithmati.model.JsonSerde;
import org.apache.kafka.common.serialization.Serde;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.Topology;
import org.apache.kafka.streams.kstream.*;

public class BankBalanceTopology {

   public static final String BANK_TRANSACTIONS = "bank-transactions";
   public static final String BANK_BALANCES = "bank-balances";
   public static final String REJECTED_TRANSACTIONS = "rejected-transactions";
```

```
    public static Topology buildTopology() {
        Serde<BankTransaction> bankTransactionSerdes = new
JsonSerde<>(BankTransaction.class);
        Serde<BankBalance> bankBalanceSerde = new JsonSerde<>(BankBalance.class);
        StreamsBuilder streamsBuilder = new StreamsBuilder();

        KStream<Long, BankBalance> bankBalancesStream =
streamsBuilder.stream(BANK_TRANSACTIONS,
                Consumed.with(Serdes.Long(), bankTransactionSerdes))
                .groupByKey()
                .aggregate(BankBalance::new,
                        (key, value, aggregate) -> aggregate.process(value),
                        Materialized.with(Serdes.Long(), bankBalanceSerde))
                .toStream();
        bankBalancesStream
                .to(BANK_BALANCES, Produced.with(Serdes.Long(), bankBalanceSerde));

        bankBalancesStream
                .mapValues((readOnlyKey, value) -> value.getLatestTransaction())
                .filter((key, value) -> value.state ==
BankTransaction.BankTransactionState.REJECTED)
                .to(REJECTED_TRANSACTIONS, Produced.with(Serdes.Long(),
bankTransactionSerdes));

        return streamsBuilder.build();
    }
}
```

5. BankBalanceApp

```
import com.github.programmingwithmati.config.StreamConfiguration;
import com.github.programmingwithmati.topology.BankBalanceTopology;
import org.apache.kafka.streams.KafkaStreams;

public class BankBalanceApp {

    public static void main(String[] args) {
        var configuration = StreamConfiguration.getConfiguration();
        // créer la topologie
        var topology = BankBalanceTopology.buildTopology();
        // créer un stream qui prend en entrées la topologie et la configuration
        var kafkaStreams = new KafkaStreams(topology, configuration);
        // démarrer le stream
        kafkaStreams.start();
        // fermer le stream s'il n'est pas utilisé
        Runtime.getRuntime().addShutdownHook(new Thread(kafkaStreams::close));
```

```
    }
}
```

4. BankBalanceTopologyTest

```
import com.github.programmingwithmati.model.BankBalance;
import com.github.programmingwithmati.model.BankTransaction;
import com.github.programmingwithmati.model.JsonSerde;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.TestInputTopic;
import org.apache.kafka.streams.TestOutputTopic;
import org.apache.kafka.streams.TopologyTestDriver;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.math.BigDecimal;
import java.util.Date;
import java.util.List;
import java.util.Properties;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;

class BankBalanceTopologyTest {

    TopologyTestDriver testDriver;
    private TestInputTopic<Long, BankTransaction> bankTransactionTopic;
    private TestOutputTopic<Long, BankBalance> bankBalanceTopic;
    private TestOutputTopic<Long, BankTransaction> rejectedBankTransactionTopic;

    @BeforeEach
    void setup() {
        Properties props = new Properties();
        props.put(StreamsConfig.APPLICATION_ID_CONFIG, "test");
        props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "dummy:1234");
        testDriver = new TopologyTestDriver(BankBalanceTopology.buildTopology(), props);

        var bankBalanceJsonSerde = new JsonSerde<>(BankBalance.class);
        var bankTransactionJsonSerde = new JsonSerde<>(BankTransaction.class);

        bankTransactionTopic =
testDriver.createInputTopic(BankBalanceTopology.BANK_TRANSACTIONS,
Serdes.Long().serializer(), bankTransactionJsonSerde.serializer());

        bankBalanceTopic =
testDriver.createOutputTopic(BankBalanceTopology.BANK_BALANCES,
```

```
Serdes.Long().deserializer(), bankBalanceJsonSerde.deserializer());
        rejectedBankTransactionTopic =
testDriver.createOutputTopic(BankBalanceTopology.REJECTED_TRANSACTIONS,
Serdes.Long().deserializer(), bankTransactionJsonSerde.deserializer());
    }

    @AfterEach
    void teardown() {
        testDriver.close();
    }

    @Test
    void testTopology() {
        List.of(
                BankTransaction.builder()
                        .balanceId(1L)
                        .time(new Date())
                        .amount(new BigDecimal(500))
                        .build(),
                BankTransaction.builder()
                        .balanceId(2L)
                        .time(new Date())
                        .amount(new BigDecimal(3000)).build(),
                BankTransaction.builder()
                        .balanceId(1L)
                        .time(new Date())
                        .amount(new BigDecimal(500)).build()
        )
                .forEach(bankTransaction ->
bankTransactionTopic.pipeInput(bankTransaction.getBalanceId(), bankTransaction));

        var firstBalance = bankBalanceTopic.readValue();

        assertEquals(1L, firstBalance.getId());
        assertEquals(new BigDecimal(500), firstBalance.getAmount());

        var secondBalance = bankBalanceTopic.readValue();

        assertEquals(2L, secondBalance.getId());
        assertEquals(new BigDecimal(3000), secondBalance.getAmount());


        var thirdBalance = bankBalanceTopic.readValue();

        assertEquals(1L, thirdBalance.getId());
        assertEquals(new BigDecimal(1000), thirdBalance.getAmount());
```

```
        assertTrue(rejectedBankTransactionTopic.isEmpty());
    }

    @Test
    void testTopologyWhenRejection() {
        List.of(
                BankTransaction.builder()
                        .id(1L)
                        .balanceId(1L)
                        .time(new Date())
                        .amount(new BigDecimal(-500))
                        .build(),
                BankTransaction.builder()
                        .id(2L)
                        .balanceId(2L)
                        .time(new Date())
                        .amount(new BigDecimal(3000)).build(),
                BankTransaction.builder()
                        .id(3L)
                        .balanceId(1L)
                        .time(new Date())
                        .amount(new BigDecimal(500)).build()
        )
                .forEach(bankTransaction ->
bankTransactionTopic.pipeInput(bankTransaction.getBalanceId(), bankTransaction));

        var firstBalance = bankBalanceTopic.readValue();

        assertEquals(1L, firstBalance.getId());
        assertEquals(new BigDecimal(0), firstBalance.getAmount());

        var secondBalance = bankBalanceTopic.readValue();

        assertEquals(2L, secondBalance.getId());
        assertEquals(new BigDecimal(3000), secondBalance.getAmount());


        var thirdBalance = bankBalanceTopic.readValue();

        assertEquals(1L, thirdBalance.getId());
        assertEquals(new BigDecimal(500), thirdBalance.getAmount());

        var bankTransaction = rejectedBankTransactionTopic.readValue();

        assertEquals(1L, bankTransaction.getId());
        assertEquals(BankTransaction.BankTransactionState.REJECTED,
bankTransaction.getState());
```

```
    }
}
```