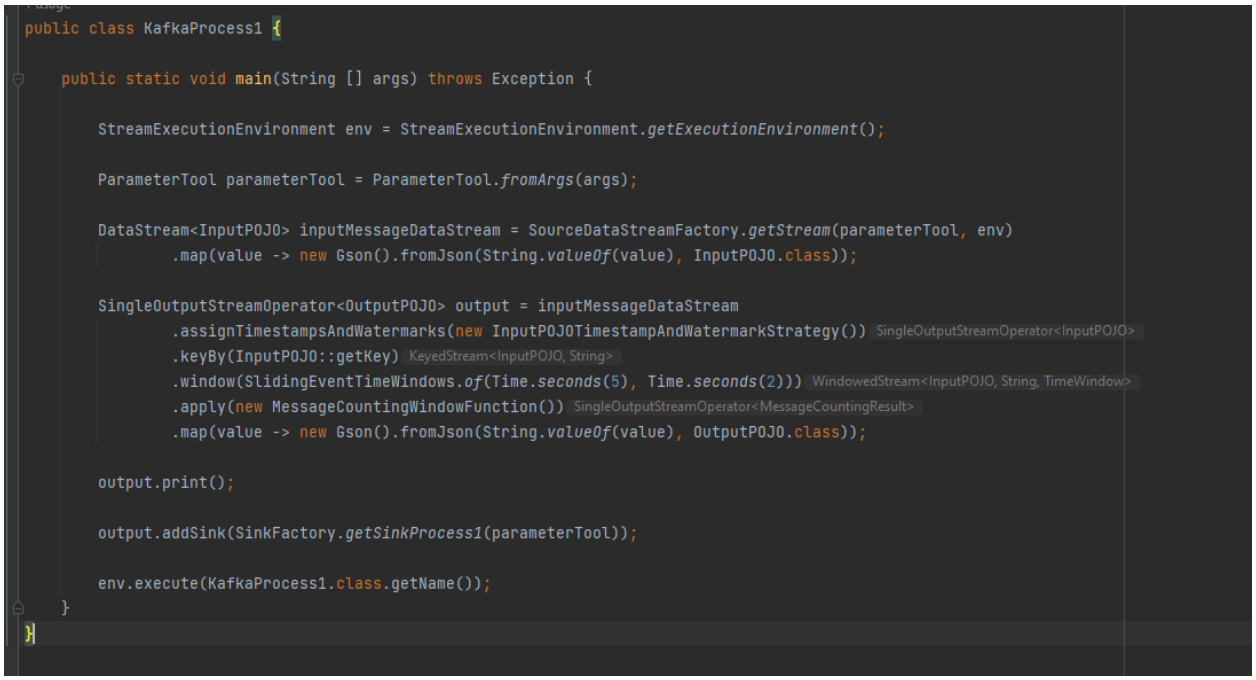
Домашна задача 1 – Flink

Милан Тасевски, 196001

1.

За решение го користам концептот опишан во курсот: „ Процесирање со ограничено доцнење на проточни податоци со Apache Flink”.

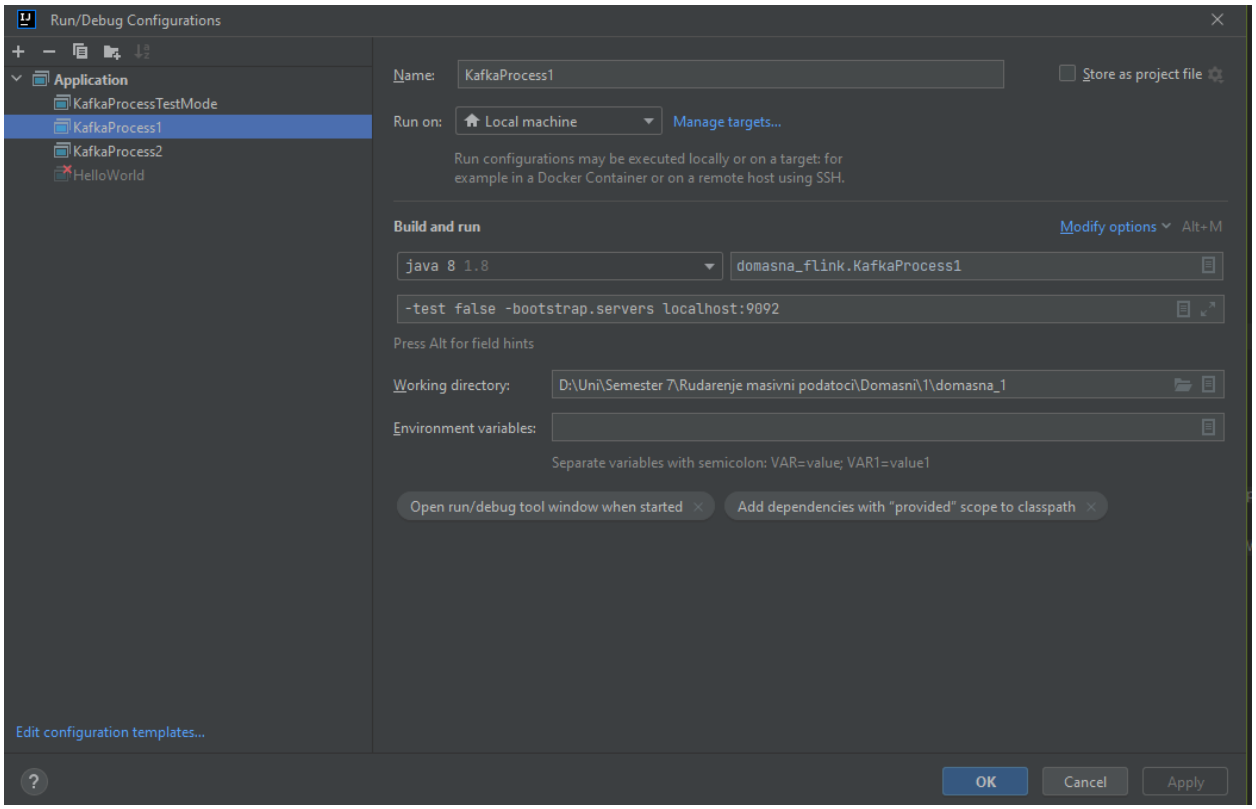


```
public class KafkaProcess1 {

    public static void main(String [] args) throws Exception {

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        ParameterTool parameterTool = ParameterTool.fromArgs(args);

        DataStream<InputPOJO> inputMessageDataStream = SourceDataStreamFactory.getStream(parameterTool, env)
                .map(value -> new Gson().fromJson(String.valueOf(value), InputPOJO.class));

        SingleOutputStreamOperator<OutputPOJO> output = inputMessageDataStream
                .assignTimestampsAndWatermarks(new InputPOJOTimestampAndWatermarkStrategy()) SingleOutputStreamOperator<InputPOJO>
                .keyBy(InputPOJO::getKey) KeyedStream<InputPOJO, String>
                .window(SlidingEventTimeWindows.of(Time.seconds(5), Time.seconds(2))) WindowedStream<InputPOJO, String, TimeWindow>
                .apply(new MessageCountingWindowFunction()) SingleOutputStreamOperator<MessageCountingResult>
                .map(value -> new Gson().fromJson(String.valueOf(value), OutputPOJO.class));

        output.print();

        output.addSink(SinkFactory.getSinkProcess1(parameterTool));

        env.execute(KafkaProcess1.class.getName());
    }
}
```

Main функцијата за првата задача е во класата KafkaProcess1.

Првин девинирам StreamExecutionEnvironment на стандардниот начин.

Користам parameterTool со зададени аргументи за извршување кои се дефинираат на следниот начин:



Секоја од пораките во inputMessageDataStream ја мапирам од објект од тип inputMessage, со едно дефинирано поле од тип JSONObject во custom POJO од тип InputPOJO кој го запазува форматот на пораките.

На вака дефинираниот stream користејќи ја InputPOJOTimestampAndWatermarkStrategy на елементите им задавам timestamp пред да ги групирам по keyBy. После тоа го дефинирам лизгачкиот прозорец со преклопувачки прозорец со големина 5s и лизгање 2s, па ја применувам со apply фунцијата MessageCountingWindowFunction која го опреретува методот apply и има за цел да брои стигнати пораки по клуч, користејќи MessageCountingAccumulator и резултатот запишувајќи го во MessageCountingResult. Ја мапирам секоја од вредностите во објект од типот OutputPOJO.

Го печатам output-от пред да го испратам на sink. Sink-от го дефинирам преку класата SinkFactory и за првата задача го користам методот getSinkProcess1. Доколку mode-от не е за тестирање, враќам FlinkKafkaProducer преку класата KafkaSinkStream користејќи ја

OutputPojoSerializationSchema и резултатот го враќам на topic дефиниран во ProjectSettings, во случајот на topic-от со име results1.

2.

```java
// Zadaca 2:
1 usage
public class KafkaProcess2 {

    public static void main(String [] args) throws Exception {

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        ParameterTool parameterTool = ParameterTool.fromArgs(args);

        DataStream<InputPOJO> inputMessageDataStream = SourceDataStreamFactory.getStream(parameterTool, env)
                .map(value -> new Gson().fromJson(String.valueOf(value), InputPOJO.class));
        // inputMessageDataStream.print();

        SingleOutputStreamOperator<String> output = inputMessageDataStream
                .assignTimestampsAndWatermarks(new InputPOJOTimestampAndWatermarkStrategy()) SingleOutputStreamOperator<InputPOJO>
                .keyBy(InputPOJO::getKey) KeyedStream<InputPOJO, String>
                .window(SlidingEventTimeWindows.of(Time.seconds(5), Time.seconds(2))) WindowedStream<InputPOJO, String, TimeWindow>
                .apply(new MessageStatsWindowFunction());

        output.print();

        output.addSink(SinkFactory.getSinkProcess2(parameterTool));

        env.execute(KafkaProcess2.class.getName());

    }
}
```

Решението на втората задача во голема мера е исто. Разликата е во неколку работи:

- Резултатот назад се враќа во String формат
- Во apply користам MessageStatsWindowFunction која овој пат освен броење, преку објект од типот DoubleSummaryStatistics врши целостна статистичка анализа според условите на задачата
- Се користи getSinkProcess2 за враќање на резултатот, кој се враќа на topic-от со име results2 преку друга функција во SinkFactory и со користење на StringSerializationSchema дефинирана од моја страна.