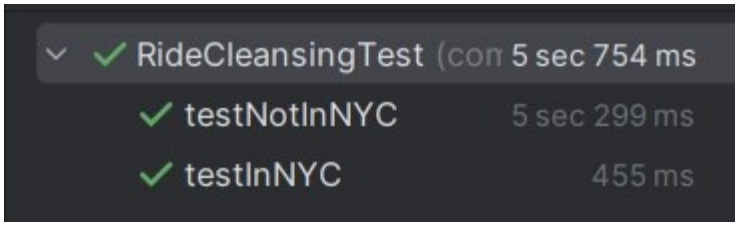
Задания выполнены на яп Java

1. RideCleanisingExercise

Фильтр написан на основе isInNYC() из пакета GeoUtils. Если начальная и конечная точки в границе NYC, фильтр возвращает true, иначе false.

```
return GeoUtils.isInNYC(taxiRide.startLon, taxiRide.startLat) &&
GeoUtils.isInNYC(taxiRide.endLon, taxiRide.endLat);
```

Скрин прохождения теста



2. RidesAndFaresExercise

Open() инициализирует переменные состояния rideState и fareState.

```
ValueStateDescriptor<TaxiRide> rideDescriptor = new
ValueStateDescriptor<>("rideState", TaxiRide.class);
ValueStateDescriptor<TaxiFare> fareDescriptor = new
ValueStateDescriptor<>("fareState", TaxiFare.class);

rideState = getRuntimeContext().getState(rideDescriptor);
fareState = getRuntimeContext().getState(fareDescriptor);
```

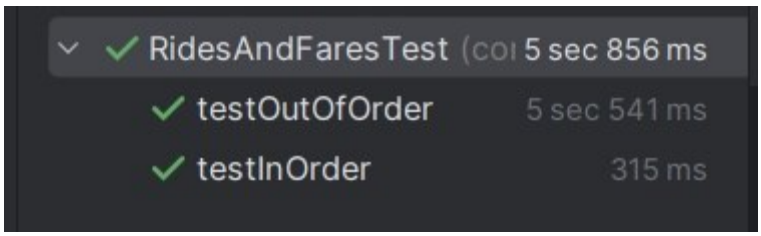
flatMap1() когда встречает событие TaxiRide, сохраняется в rideState. Если в fareState уже есть TaxiFare, то генерируется новый Tuple (ride, fareState.value()), fareState очищается.

```
rideState.update(ride);
if (fareState.value() != null) {
   out.collect(new Tuple2<>(ride, fareState.value()));
   fareState.clear();
}
```

flatMap2() когда встречает событие TaxiFare, сохраняется в fareState. Если в rideState уже есть TaxiRide, то генерируется новый Tuple (rideState.value(), fare), rideState очищается.

```
fareState.update(fare);
if (rideState.value() != null) {
   out.collect(new Tuple2<>(rideState.value(), fare));
   rideState.clear();
}
```

Скрин прохождения теста

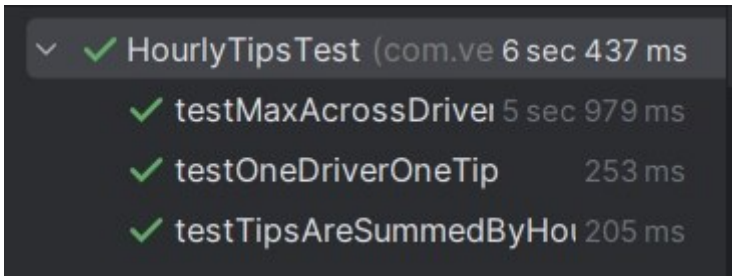


3. HourlyTipsExerxise

Группируем по id водителя TaxiFare, группируем данные по каждому часу для каждого водителя, перегруппируем при помощи timeWindowAll, для каждого часа находим наибольшую заработанную сумму.

```
DataStream<?> hourlyMax = fares
      .keyBy((TaxiFare fare) -> fare.driverId)
      .timeWindow(Time.hours(1))
      .process(new ProcessWindowFunction
            <TaxiFare, Tuple3<Long, Long, Float>, Long, TimeWindow> () {
         @Override
         public void process(Long key,
                        Context context,
                        Iterable<TaxiFare> fares,
                        Collector<Tuple3<Long, Long, Float>> out) {
            Float sum = 0.F;
            for (TaxiFare fs : fares)
               sum += fs.tip;
            out.collect(new Tuple3<>(context.window().getEnd(), key, sum));
         }
      })
      .timeWindowAll(Time.hours(1))
      .maxBy(2);
```

Скрин прохождения теста



4. ExpiringStateExercise

Open() инициализирует переменные состояния rideState и fareState.

```
ValueStateDescriptor<TaxiRide> rideStateDescriptor = new
ValueStateDescriptor<>("rideState", TaxiRide.class);
ValueStateDescriptor<TaxiFare> fareStateDescriptor = new
ValueStateDescriptor<>("fareState", TaxiFare.class);

rideState = getRuntimeContext().getState(rideStateDescriptor);
fareState = getRuntimeContext().getState(fareStateDescriptor);
```

onTimer() Таймер, зарегистрированный в processElement1() или processElement2(), срабатывает, когда в течении указанного времени не поступило ни одно подходящее событие. При срабатывании таймера проверяется fareState и rideState для определения несогласованных событий. Если несогласованные данные найдены, то они передаются в unmatchedFares или unmatchedRides и состояние отчищается.

```
if (fareState.value() != null) {
   ctx.output(unmatchedFares, fareState.value());
   fareState.clear();
}
if (rideState.value() != null) {
   ctx.output(unmatchedRides, rideState.value());
   rideState.clear();
}
```

processElement1() Если соответствующий ride TaxiFare существует, создается новый кортеж (ride, currentFare), состояние TaxiFare очищается. Если соответствующий TaxiFare не найден, текущий TaxiRide сохраняется в rideState, и регистрируется таймер.

```
TaxiFare currentFare = fareState.value();
if (currentFare != null) {
   out.collect(new Tuple2<>(ride, currentFare));
   fareState.clear();
}
else {
   rideState.update(ride);
   context.timerService().registerEventTimeTimer(ride.getEventTime());
}
```

processElement2() Если соответствующий fare TaxiRide существует, создается новый кортеж (currentRide, fare), состояние TaxiRide очищается. Если соответствующий TaxiRide не найден, текущий TaxiFare сохраняется в fareState, и регистрируется таймер.

```
TaxiRide currentRide = rideState.value();
if (currentRide != null) {
   out.collect(new Tuple2<>(currentRide, fare));
   rideState.clear();
}
else {
   fareState.update(fare);
   context.timerService().registerEventTimeTimer(fare.getEventTime());
}
```

Скрин прохождения теста