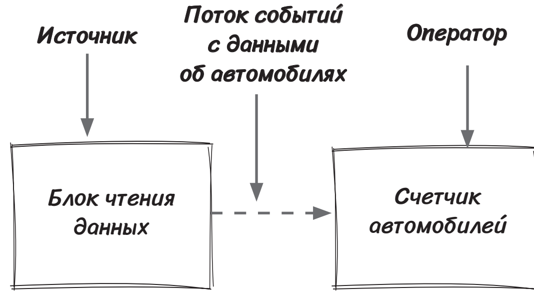
Последовательность выполнения стримингового задания

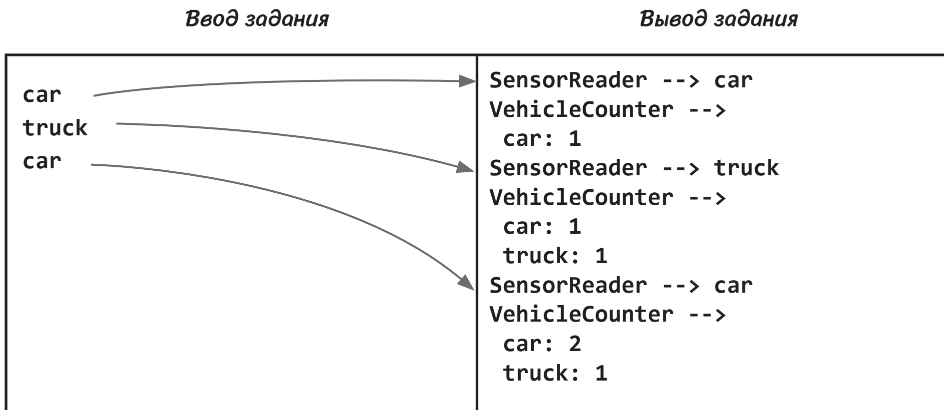
С концепциями, рассмотренными на двух предыдущих страницах, можно составить визуализацию стримингового задания для подсчета автомобилей, состоящего из двух компонентов и одного потока между ними, как показано справа.



- Блок чтения данных получает данные с датчика и сохраняет события в очереди. Это источник.
- Счетчик автомобилей отвечает за подсчет автомобилей в потоке. Это оператор.
- Непрерывное перемещение данных от источника к оператору — поток событий автомобилей.

Блок чтения данных с датчика становится началом задания, а счетчик автомобилей — его концом. Линия, соединяющая блок чтения (источник) со счетчиком автомобилей (оператором), представляет поток типов автомобилей (событий), проходящий от блока чтения данных к счетчику автомобилей.

В этой главе такая система будет описана более подробно. Она будет выполняться на ваших локальных компьютерах с двумя терминалами: один получает ввод пользователя (левый столбец), а другой — вывод задания (правый столбец).



Первое стриминговое задание

Стриминговое задание создается средствами Streamwork API и включает следующие шаги.

1. Создание класса задания.
2. Построение источника.
3. Построение оператора.
4. Соединение компонентов.

Первое стриминговое задание: создание класса события

Событие — один фрагмент данных в потоке, который должен обрабатываться заданием. Во фреймворке Streamwork класс API `Event` отвечает за хранение или инкапсуляцию пользовательских данных. Аналогичные концепции существуют и в других стриминговых системах.

В задании каждое событие представляет один тип автомобилей. Для простоты будем считать, что каждый тип представляет собой строку (например, `car` или `truck`). В нашем примере будет использоваться класс события `VehicleEvent`, производный от класса `Event` из API. Каждый объект `VehicleEvent` содержит информацию об автомобиле, которую можно получить вызовом функции `getData()`.

Внутренняя строка для обозначения типа автомобиля.

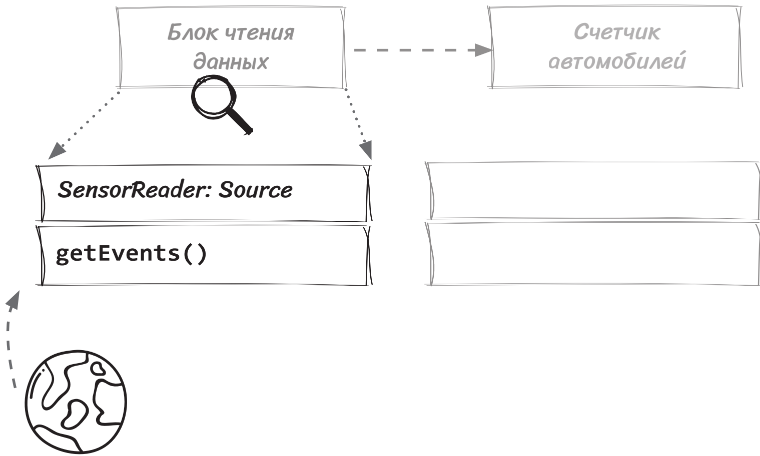
```
public class VehicleEvent extends Event {
  private final String vehicle;

  public VehicleEvent(String vehicle) {
    this.vehicle = vehicle;
  }

  @Override
  public String getData() {
    return vehicle;
  }
}
```

Конструктор получает vehicle в виде строки и сохраняет значение.

Получает данные, хранящиеся в событии.

Первое стриминговое задание: источник данных

Источником (source) называется компонент, который вводит внешние данные в стриминговую систему. Земной шар на следующей диаграмме обозначает данные, внешние по отношению к заданию. В стриминговом задании блок чтения данных от датчика вводит данные, полученные от локального порта, в систему.

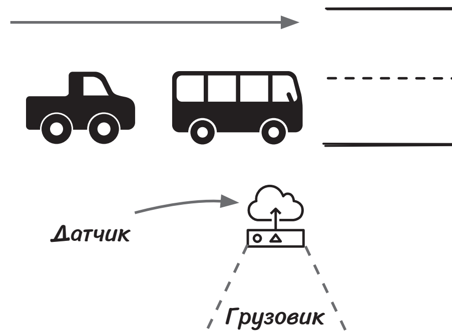


Во всех стриминговых фреймворках присутствует API, дающий возможность задавать для источников данных логику, которая представляет интерес только для вас. Во всех API источников данных присутствует некая разновидность *перехватчика жизненного цикла* (lifecycle hook), который будет вызываться для получения внешних данных. В этой точке код выполняется фреймворком.

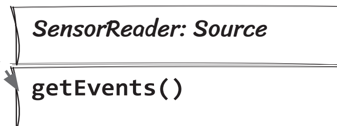
> ***Что такое перехватчик жизненного цикла?***
>
> Перехватчиками жизненного цикла называются методы, которые вызываются по определенному повторяемому шаблону фреймворком, которому они принадлежат. Как правило, эти методы позволяют разработчикам настроить поведение приложения в фазах жизненного цикла фреймворка, на основе которого строится приложение. В случае фреймворка Streamwork имеется перехватчик жизненного цикла (или метод), называемый getEvents(). Он многократно вызывается фреймворком, чтобы вы могли получить внешние данные. Перехватчики жизненного цикла позволяют разработчикам писать логику, которая для них важна, и поручить рутинную работу фреймворку.

Первое стриминговое задание: источник данных (продолжение)

В этом задании блок чтения данных датчика будет читать события. В нашем упражнении вы будете моделировать датчик на мосту, самостоятельно создавая события и передавая их на открытый порт вашего компьютера, прослушиваемый стриминговым заданием. Блок чтения получает данные о типах автомобилей, отправленные в порт, и передает их потоковому заданию — так выглядит бесконечный (или неограниченный) поток событий.



1. getEvents() содержит логику чтения данных от датчика, определяемую пользователем.



2. События направляются в исходящий поток (очередь).

Java-код класса SensorReader выглядит так:

Перехватчик жизненного цикла стриминговой системы выполняет логику, определяемую пользователем.

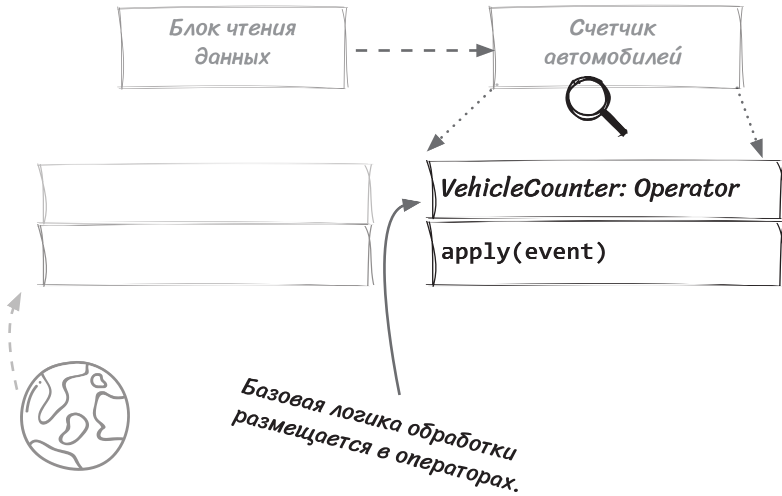
Чтение типа автомобиля из входных данных.

```
public class SensorReader extends Source {
  private final BufferedReader reader;
  public SensorReader(String name, int port) {
    super(name);
    reader = setupSocketReader(port);
  }

  @Override
  public void getEvents(List<Event> eventCollector) {
    String vehicle = reader.readLine();
    eventCollector.add(new VehicleEvent(vehicle));
    System.out.println("SensorReader --> " + vehicle);
  }
}
```

Передача строки получателю.

Первое стриминговое задание: оператор

Вся пользовательская логика обработки содержится в операторах. Они отвечают за получение входящих событий для обработки и генерирование исходящих событий; следовательно, у них есть как ввод, так и вывод. Вся логика обработки данных в стриминговых системах обычно размещается в компонентах операторов.



Для простоты мы ограничились одним источником и одним оператором. Рассматриваемая реализация счетчика автомобилей только подсчитывает автомобили, а затем регистрирует текущее значение счетчика в системе. Другой (возможно, лучший) способ реализации системы основан на направлении данных в новый поток. Тогда регистрация результатов может выполняться в дополнительном компоненте, который следует за счетчиком автомобилей. Как правило, в задании используются компоненты, которые имеют только одну функцию.

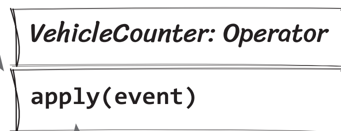
Кстати говоря, Сид занимает должность технического директора. Иногда он бывает старомодным, но он очень умен и интересуется новыми технологиями.

Первое стриминговое задание: оператор (продолжение)

В компоненте VehicleCounter карта <vehicle, count> используется для хранения счетчиков типов автомобилей в памяти. Она обновляется соответствующим образом при получении нового события. В стриминговом задании счетчик автомобилей представляет собой оператор, который подсчитывает события. Этот оператор завершает задание и не генерирует вывод для последующих операторов.

Ключ (Vehicle)	*Значение (Count)*
автомобиль	*2*
грузовик	*1*
микроавтобус	*1*

1. Получение входящих событий.



2. Логика, определяемая пользователем, применяется к событиям данных вызовом apply().

```
public class VehicleCounter extends Operator {
  private final Map<String, Integer> countMap =
    new HashMap<String, Integer>();

  public VehicleCounter(String name) {
    super(name);
  }

  @Override
  public void apply(Event event,List<Event> collector) {
    String vehicle = ((VehicleEvent)event).getData();
    Integer count = countMap.getOrDefault(vehicle, 0);
    count += 1;
    countMap.put(vehicle, count);
    System.out.println("VehicleCounter --> ");
    printCountMap();
  }
}
```

Получение счетчиков из карты

Увеличение счетчика

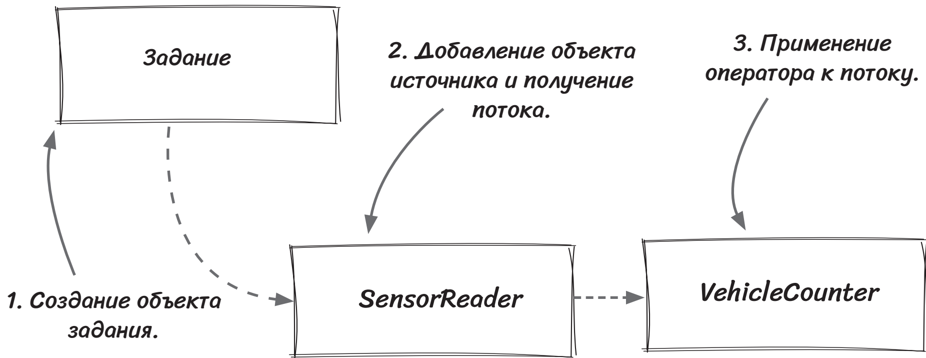
Сохранение счетчика обратно в карте

Вывод текущего значения счетчика

Первое стриминговое задание: сборка задания

Чтобы собрать стриминговое задание, необходимо добавить источник `SensorReader` и оператор `VehicleCounter` и соединить их. В классах `Job` и `Stream`, которые мы построили для вас, присутствуют перехватчики:

- `Job.addSource()` добавляет источник данных в задание.
- `Stream.applyOperator()` добавляет оператор в поток.



Следующий код выполняет описанные выше шаги:

```
public static void main(String[] args) {
  Job job = new Job();                                   ← Создание объекта задания
  Stream bridgeOut=job.addSource(new SensorReader()); ←
                     Добавление объекта потока и получение потока

  bridgeOut.applyOperator(newVehicleCounter());   ← Применение оператора
  JobStarter starter = new JobStarter(job);         к потоку
  starter.start();            ← Запуск задания
}
```