Exercise 03 - Handling `New Product` Events

Required Services

The following services are involved and have to be started before the final exercise validation:

- NotificationSrv (`http://localhost:8010`)
- OrderProcessSrv (`http://localhost:8020`)
- ProductSrv (`http://localhost:8050`)
- WarehouseSrv (`http://localhost:8070`)
- Apache Zookeeper (`localhost:2181`, starts automatically in the provided Ubuntu VM)
- Apache Kafka (`localhost:9092`, starts automatically in the provided Ubuntu VM)

Description

The product management team has decided to establish a new follow-up process when a new product has been added to the DB of the `ProductSrv`. To make their job easier and reduce manual efforts, several automatic actions have to be performed when a new product has been successfully created via `POST http://localhost:8050/products (experiment.webshop.products.resources.ProductResource)`.

To handle this kind of event-based messaging, the lead developer decided to use the message-oriented middleware Apache Kafka. Kafka provides so called `topics` that message producers can publish to. Message consumers can subscribe to `topics` and will receive all published messages. This decouples producers and consumers and provides reliable asynchronous communication. The basic infrastructure for this is already in place. The `ProductSrv` already publishes a `new product event` (which is basically an instance of `experiment.webshop.products.api.Product`) to the Kafka topic `new-products`. The logic for this is located in `experiment.webshop.products.messaging.KafkaNotifier` and does not have to be changed. Similarly, the `NotificationSrv` and the `WarehouseSrv` already subscribe to these events (see the classes `experiment.webshop.notifications.messaging.KafkaListener` and `experiment.webshop.warehouse.messaging.KafkaListener`), but currently do nothing when such an event comes in.

Tasks

The following actions have to be performed in response to a `new product event`:

1. **NotificationSrv: Add the product to the internal new product DB.** The `NotificationSrv` has an internal DB where new products have to be stored. This action has to be implemented in the `run()` method of the `experiment.webshop.notifications.messaging.KafkaListener` class. Use the provided `notificationResource` to invoke its `addNewProduct()` method. The payload for this method is the newly created `experiment.webshop.notifications.api.Product` instance that is part of the Kafka message.
2. **NotificationSrv: Notify the sales department about the new product.** The `NotificationSrv` provides functionality for sending a mail to the sales department. This action has to be implemented in the `run()` method of the `experiment.webshop.notifications.messaging.KafkaListener` class, in the same fashion as for task 1. Use the provided `notificationResource` to invoke its `sendProductMail()` method. The payload for this method is an instance of

`experiment.webshop.products.api.NewProductMailRequest`. Below is an exemplary payload (`product` will of course be the newly created product):

```
{
    "type": "NEW_PRODUCT_MAIL",
    "product": {
        "id": 1,
        "name": "NewTestProduct"
        "categoryId": 1,
        "price": 9.99
    }
}
```

3. **WarehouseSrv: Stock-up on 10 copies of the newly created product.** As a start, the `WarehouseSrv` needs to have 10 copies of the new product available for purchase. This action has to be implemented in the `run()` method of the `experiment.webshop.warehouse.messaging.KafkaListener` class, in the same fashion as for task 1 and 2. To initiate the stock-up process, use the provided `warehouseResource` to invoke its `updateProductAvailability()` method. Since this method requires the `productId` as a `io.dropwizard.jersey.params.LongParam.LongParam` and the `amount` as an `io.dropwizard.jersey.params.IntParam.IntParam`, you need to convert the values before using them as parameters:

```
LongParam productId = new LongParam(String.valueOf(createdProduct.getId()));
IntParam amount = new IntParam("10");
```

Validation

When you are finished with all tasks, make sure all required services (see Required Services) and the exercise validation UI is up and running (if not, execute `exercise-validation/build-and-run-validation-ui.sh`) and then navigate to `http://localhost:5001` (**it is important to start from this page, because it will determine which version you are working on**). Click on `Exercise 03` and then on `Start Validation`. If every check is successful (`status: true`), pause your stopwatch and notify an experiment admin to write down your time.