


# Занятие 5

## Задания

### Самостоятельное изучение

- [Что такое Spring Framework? От внедрения зависимостей до Web MVC](#)
-  Евгений Борисов — Spring-потрошитель, часть 1
- Разобраться в нюансах @Autowired. Когда производится по типу, имени, квалификатору. @Primary
- Разобрать, как вызвать метод при старте приложения. @PostConstruct, DisposableBean, Spring Event
- Разобрать жизненный цикл бина в Application Context

# 1.Event Sourcing

Задача из предыдущего ДЗ. Должно быть реализована с разбиением на Spring компоненты, а потом запуском как spring приложения. В качестве кандидатов на оформление в виде компонентов - DataSource, ConnectionFactory, классы для отправки сообщений, классы для получения сообщений, работы с БД

## 2. Query Extender

Делая запрос к метаданным БД, необходимо узнать, какие колонки есть в конкретной таблице и, используя эти данные, написать корректный SQL запрос к этой таблице. Работа с метаданными должна производиться при помощи объекта **java.sql.DatabaseMetaData**

Написать Spring Component, реализующий интерфейс

```
public interface SQLQueryBuilder {  
    String queryForTable(String tableName);  
    List<String> getTables();  
}
```

Метод **queryForTable** получает на вход имя таблицы и выполняет следующее

1. Проверяет, что данная таблица есть в БД
2. Если таблицы нет - метод возвращает null
3. Если таблица есть - получает список колонок
4. На основании списка колонок составляется строка запроса вида  
"SELECT <col1>, <col2>, <col3> FROM <tablename>"
5. Данная строка возвращается в качестве результата выполнения метода

Пример: в БД есть таблица **person** с колонками id, first\_name, last\_name, middle\_name. В результате метод должен вернуть строку

```
SELECT id, first_name, last_name, middle_name FROM person
```

Метод **getTables** возвращает в качестве результата список имен всех таблиц, которые есть в БД

### 3.Message Filter (18+)

Необходимо написать приложение, которое бы осуществляло цензуру присылаемых сообщений

1. Приложение получает сообщение из очереди input.
2. Сообщение - это строка, в котором хранится предложение
3. Приложение проверяет каждое слово полученного предложения на предмет полного совпадения с одним из слов из списка (без учета регистра). Слово - это последовательность символов, Список находится после этого абзаца, но цвет букв - белый. Выделите белое пространство внизу для получения доступа к тексту
4. Если слово совпадает с каким-либо словом из списка - то все буквы в в этом слове, кроме первой и последней заменяются на \*
5. Если слово не совпадает - оно остается без изменений
6. Модифицированное таким образом предложение отправляется в очередь output

Пример:

Получено из очереди input: **Fuck you, уважаемый!**

Отправлено в очередь output: **F\*\*k you, уважаемый!**

#### Примечания по реализации

1. Очереди **input** и **output** - очереди RabbitMQ
2. Недопустимые слова должны быть сохранены в файле построчно, в каждой строке одно слово
3. Слово в исходном сообщении - это последовательность символов, ограниченная символами пробела/точкой/запятой/точкой с запятой/вопросительным знаком/восклицательным знаком/концом строки.

4. Таблица со списком нецензурных слов должна создаваться при запуске приложения. Но только в том случае, если такой таблицы нет в БД. (Изучить и использовать для проверки существования метод `Connection.getMetaData()`)
5. Список нецензурных слов необходимо поместить в файл, а затем, при запуске приложения, переместить данные в БД. При каждом новом запуске необходимо очищать таблицу в БД и записывать данные из файла снова
6. При обработке необходимо делать запрос в БД для проверки нахождения слова в списке
7. Приложение должно быть написано с использованием Spring Framework; должны быть определены и реализованы необходимые компоненты
8. Для проверки можно использовать RabbitMQ management console для отправки сообщений в очередь input
9. Для сборки проекта необходимо использовать Maven
10. DataSource должен создаваться со следующими параметрами подключения

```
PGSimpleDataSource dataSource = new PGSimpleDataSource();
dataSource.setServerName("localhost");
dataSource.setPortNumber(5432);
dataSource.setDatabaseName("postgres");
dataSource.setUser("postgres");
dataSource.setPassword("postgres");
```