

SummaryTask3

На странице представлена информация по заданию SummaryTask3

❖ Формулировка бизнес области поставлена каждому в Jenkins индивидуально (SummaryTask3).

❖ Пример можно забрать здесь: [ST3Example](#)

- [Задание](#)
- [Пример](#)
 - [ST3Example](#)
 - [ST3ExampleExtra](#)
- [Замечание](#)

❖ [input.xsd](#) должен декларировать целевое пространство имен (атрибут `targetNamespace` должен присутствовать)

[input.xml](#) должен быть валидным относительно [input.xsd](#)

Выходные XML документы [output.dom.xml](#), [output.sax.xml](#) и [output.stax.xml](#) должны быть валидны относительно исходной XML схемы [input.xsd](#) если из нее исключить декларирование целевого пространства имен (см. файл `input-no-targetNamespace.xsd` в `ST3Example`)

(Замечание: выходные XML документы не обязаны объявлять атрибут `noNamespaceSchemaLocation`)

Задание

Задача сформулирована следующим образом.

1. Создать файл XML (`input.xml`) и соответствующую ему схему XSD (`input.xsd`).
2. При разработке XSD обязательно использовать: простые и комплексные типы, перечисления, шаблоны и предельные значения.
3. Создать Java-класс, соответствующий данному описанию.
4. Создать Java-приложение (назвать `Main`) для разбора XML-документа и инициализации контейнера объектов информацией из XML-файла. Для разбора использовать: SAX, StAX парсеры, а также DOM анализатор (все три варианта).
5. Определить методы, которые будут сортировать объекты контейнера с использованием интерфейса `Comparator` по некоторому параметру или набору параметров (три парсера `>`; три варианта сортировки `>`; три метода).
6. Произвести проверку XML-документа на валидность относительно XSD (с помощью валидирующего парсера или `validation API`).
7. Определить метод сохранения информации из контейнера в XML-документ.

8. Продемонстрировать работу приложения:
 - каждым из парсеров прочитать XML-документ;
 - получить контейнер объектов;
 - отсортировать объекты в контейнере (одним из трех способов);
 - сохранить отсортированный контейнер в XML-документ (в файл).
9. Входными данными приложения являются имена двух файлов: XML-документа и XSD-документа; выходными данными - три XML файла. Имена входных файлов задавать параметрами командной строки (два параметра или один, если валидирование будет происходить относительно схемы, заданной внутри XML-документа).
10. Для демонстрации работы приложения создать класс Demo, который вызывает метод Main.main вашего приложения с соответствующим значением параметров командной строки (хардкод параметров командной строки):


Demo.java

```
package ua.nure.your_last_name.SummaryTask3;
/**
 * Demo class to run project WO command line.
 */
class Demo {
    public static void main(String[] args) throws Exception {
        Main.main(new String[] { "input.xml" });
    }
}
```

11. ~~Написать тесты, покрытие кода тестами должно быть 100%, тесты должны находиться в своем src каталоге с именем test.~~
12. Классы верхнего уровня и нетривиальные методы должны быть документированы.
13. Опционально. Создать XSL документ преобразующий XML-документ в документ HTML.

Соглашение об именовании
<p>Корневой элемент XML документа назвать согласно названию задачи или исходя из того что это контейнер однотипных элементов, имя которого указано в задаче.</p> <p>Пример: если задача называется Тестирование, а элемент имеет имя Test, то корневой элемент можно назвать либо Testing либо Tests.</p> <p>Входные и выходные файлы должны иметь следующие имена.</p> <p>Входные файлы:</p> <ul style="list-style-type: none">• input.xml - XML-документ, валидный относительно input.xsd• input.xsd - XSD-докумен, схема для input.xml <p>Выходные файлы (результат):</p> <ul style="list-style-type: none">• output.dom.xml - результат работы DOM-анализатора и последующей сортировки одним из трех способов.• output.sax.xml - результат работы SAX-парсера и последующей сортировки одним из трех способов.• output.stax.xml - результат работы StAX-парсера после сортировки одним из трех способов.

Пример решения аналогичной задачи находится в репозитории по адресу:
/examples/projects/ST3Example

 Далее идет словесное описание бизнес области (у каждого своя область).

Пример

В репозитории находится пример решения аналогичной задачи.

Тест.

Вопросы теста имеют следующие характеристики:


Question - содержание вопроса;


Answers - ответы (один или несколько правильных)

Корневой элемент Test.

В репозитории содержится два варианта решения задачи.

№	Описание	Адрес	Примечание
1	Содержит решение задачи с применением технологий DOM, SAX, StAX (высокоуровневый)	/examples/projects/ST3Example	Достаточно ограничить решение своей задачи указанными технологиями.
2	Вариант №1 плюс дополнительно низкоуровневый StAX, а также JDOM, JAXB, Validation API, XSLT	/examples/projects/ST3ExampleExtra	Рекомендуется (это не обязательно!) использовать указанные технологии при решении своей задачи (данный пример в репозитории отсутствует)

 Вариант №2 рекомендуется брать за основу при решении своей задачи, если вы хотите более углубленно изучить Java+XML.

 Брать за основу не означает взять код и править его под свою задачу. Создавайте проект с нуля. Пример служит исключительно для ознакомления и понимания схемы работы тех или иных технологий. Смотрите в код примера и делайте свою задачу.

ST3Example

В примере показана работа с технологиями DOM, SAX, StAX (высокоуровневый, основанный на XMLEventReader) . Подключение внешних библиотек не требуется. Но если положить в classpath внешнюю реализацию DOM/SAX/StAX то будет работать именно она.

Для каждой технологии написан свой контроллер. В каждом из них есть метод main, который тестирует соответствующий контроллер. Для запуска приложения достаточно в Eclipse запустить метод main класса Demo. Метод main класса Main демонстрирует работу всех трех контроллеров, при этом входные данные:

- input.xsd - XML схема с целевым пространством имен;
- input.xml - XML документ, валидный, относительно схемы (корневой элемент квалифицирован, т.е. с префиксом).

Каждый контроллер разбирает соответствующей технологией входной XML документ и создает контейнер Test, содержащий объекты Question. Далее метод main при помощи класса утилиты Sorter сортирует контейнер на основе каких то критериев (всего три способа сортировки - для каждого контроллера свой) и сохраняет результат в выходном XML документе. Сохранение осуществляется с помощью трансформации контейнера Test в DOM, а затем DOM сохраняется в XML (один из возможных способов).



Выходные XML документы не будут являться валидными относительно input.xsd, т.к. корневой элемент в них не квалифицирован (сделано для упрощения кода). Как квалифицировать корневой элемент - см. код проекта ST3Example.

Если собрать приложение в jar файл, то запустить можно так:

```
java -jar ST3Example.jar input.xml
```

При этом в каталоге, откуда вы запускаете приложение, должен находиться файл схема input.xsd, т.к. DOMController и SAXController валидируют документ XML относительно данной схемы.

ST3ExampleExtra

Дополнительно к предыдущему варианту добавлен контроллеры для JDOM, JAXB; STAX контроллер дополнен низкоуровневым разбором с помощью логического курсора (см. метод STAXController#parse2). Также показана работа с Validation API и работа с XSL преобразованиями.

Проект в Eclipse содержит в classpath ссылку на четыре пользовательские библиотеки: JDOM, Xerces, Woodstox и JAXB-RI. Но для запуска требуется только JDOM - нужно создать пользовательскую библиотеку с именем JDOM и подключить ее к проекту, в противном случае код не откомпилируется. Подключение остальных библиотек не обязательно. Для запуска приложения достаточно в Eclipse запустить метод main класса Demo. Метод main класса Main демонстрирует работу пяти контроллеров, при этом входные данные:

- input.xsd - XML схема с целевым пространством имен;
- input.xml - XML документ, валидный, относительно схемы (корневой элемент квалифицирован, т.е. с префиксом);
- input.xsl - XSL документ преобразования input.xml в HTML файл.

Если собрать приложение в jar файл, то запустить можно так:

```
java -jar ST3ExampleExtra.jar input.xml input.xsd input.xsl
```

При этом в jar файле в META-INF/MANIFEST.MF должна быть указана зависимость jdom:

META-INF/MANIFEST.MF

...

Class-Path: jdom-2.0.3.jar

Дополнительно можно подключить следующие библиотеки:

- Xerces - работа DOMController и SAXController будет основана на внешней реализации DOM/SAX из Xerces.
- Woodstox - работа STAXController будет основана на внешней реализации StAX из Woodstox
- JAXB-RI - JAXBController сможет контролировать имя префикса корневого элемента, если вы запускаете приложение с помощью JVM 6 (для седьмой версии подключение данной библиотеки не обязательно).

Выходные XML документы будут валидными относительно input.xsd.

В методе Main#main продемонстрировано сохранение XML тремя способами:

1. через DOM как и в ST3Example;
2. с помощью JAXB;
3. с помощью JDOM.

Замечание

Возможно, окажется полезной информация по наследованию типов в XSD: [Type inheritance](#)