**1.**

По сути XML представляет собой обычный текст, который разделяется на логические группы с помощью специальных меток, которые называют “тэг”. Назначение тэгов очень простое — надо отметить/выделить/сгруппировать какую-то информацию для того, чтобы потом ее можно было использовать. Это может быть список имен, список книг, список фирм, список вакансий и т.д.

 XML — очень мощная технология, которая позволяет хранить, передавать и обрабатывать сложно структурированные данные.

1. Разбор. Надо уметь разобрать строку на что-то более удобное для обработки — пытаться вставить внутрь строки или находить какое-то поле определенной записи из строки — это достаточно неудобно. Значит нам надо иметь некоторый набор классов для представления нашей строки в виде структуры объектов.
2. Поиск. По структуре данных надо уметь что-то искать. Причем не подстроку, а какую-то группу полей, которые относятся к определенному объекту — например полная информация о книге — наименование, авторы, отзывы. Или список контактов с фамилией “Сидоров”.
3. Проверка. Данные должны быть корректными, т.е. там должны быть только определенные поля, с определенным наполнением и они должны быть правильно скомпанованы в нашей строке.
4. Преобразование. Хоть XML достаточно удобно описывает структурированные данные, это не значит, что его удобно просматривать обычному человеку или всегда удобно обрабатывать. Нередко для решения этого вопроса требуется преобразовать XML в какое-либо другое текстовое представление — например в тот же HTML (который является частным воплощением XML). Или даже в обычный текст.

XSD — это язык описания структуры XML документа. Его также называют XML Schema. При использовании XML Schema XML парсер может проверить не только правильность синтаксиса XML документа, но также его структуру, модель содержания и типы данных.

Такой подход позволяет объектно-ориентированным языкам программирования легко создавать объекты в памяти, что, несомненно, удобнее, чем разбирать XML как обычный текстовый файл.

**2.**

Валидация – это мощный инструмент, позволяющий быстро проверить, соответствуют ли документы вашим ожиданиям и, как следствие, оперативно отклонять те из них, которые не могут быть обработаны вашим приложением.

При работе с расширяемым языком разметки (XML) валидация, как правило, сводится к написанию подробной спецификации структуры документов на одном из языков схем, каковыми являются XML Schema (XSD, стандарт консорциума WWW), RELAX NG , DTD (Document Type Definitions) и Schematron. Валидация может выполняться как непосредственно в момент разбора документа, так и позже, но обычно – перед любой обработкой его содержимого (не стоит воспринимать это утверждение как строгое правило – у него есть исключения).

До недавнего времени API (прикладной интерфейс), через который приложения выполняли валидацию, определялся языком описания схем и используемым парсером. Работа с DTD и XSD осуществлялась через опции интерфейсов SAX (Simple API for XML), DOM (Document Object Model) и JAXP (Java API for XML Processing).

* 1. Разбор — Parsing

В слэнге программистов часто используется слова “парсинг”, который и обозначает разбор строки в какую-то структуру. Здесь надо выделить два момента:

1. Разбор строки определенным алгоритмом
2. Сохранение результатов разбора в какую-то структуру вместо строки — ибо со строкой многие операции делать просто неудобно.

Что касается второго пункта, то на сегодня существует по сути одна унифицированная структура, которая называется Document Object Model (DOM).

**DOM** представляет собой набор интерфейсов (и их реализаций), которые являются специализированными объектами для хранения “узлов” (node) XML-документа. По сути каждый тэг — это “узел”. Информация внутри тэга — тоже нода. По сути любой разобранный XML — это набор элементов типа Node и еще более специализированных, построенных в дерево.

Почему дерево? Да потому что у каждой ноды может быть список дочерних нод и у каждой из них тоже может быть “детки”. Так и строится дерево. Если вам сложно это увидеть, то советую посмотреть какую-либо классическую книгу по алгоритмам и структурам данных — можно старого доброго Никлауса Вирта «Алгоритмы и структуры данных». Книгу можно найти в интернете, ну или купить.

**SAX** — это набор классов и интерфейсов, задача которых дать механизм разбора XML в строковом представлении. Основными классами я бы назвал два:

1. **SAXParser**
2. **DefaultHandler**

класс **SAXParser** начинает читать содержимое XML и когда наступает момент «Х» — начинается новый тэг, заканчивается тэг, начинается текст внутри тэга — он может вызывать определенную функцию класса **DefaultHandler**. Для того, чтобы «наполнить» эти вызовы со стороны **SAXParser** хоть каким-то смыслом, надо расширить класс **DefaultHandler**.

### **StAX**

В версии Java SE 6 появился еще один вариант разбора XML — Streaming API for XML (StAX). У этой технологии я хотел бы выделить два важных момента:

1. В отличии от SAX, который вызывает ваш обработчик тогда, когда считает нужным, в StAX вы сами управляете перемещением по тэгам
2. StAX позволяет не только читать, но и писать

**5.**

**Сериализация** — это процесс сохранения состояния объекта в последовательность байт. **Десериализация** — это процесс восстановления объекта из этих байт.

Сериализация объектов, как новая возможность введенная в JDK 1.1, предоставляет функцию для преобразования групп или отдельных объектов, в поток битов или массив байтов, для хранения или передаче по сети. И как было сказано, данный поток битов или массив байтов, можно преобразовать обратно в объекты Java. Главным образом это происходит автоматически благодаря классам ObjectInputStream и ObjectOutputStream. Программист может решить реализовать эту функцию, путем реализации интерфейса Serializable при создании класса.

[**JSON**](http://json.org/) (JavaScript Object Notation) — это легкий текстовый, независимый от языка формат обмена данными, который легко читать и писать людям и машинам. JSON может представлять два структурированных типа: объекты и массивы. Объект — это неупорядоченная коллекция из нуля или более пар имя / значение. Массив — это упорядоченная последовательность из нуля или более значений. Значения могут быть строками, числами, логическими значениями, нулем и этими двумя структурированными типами.

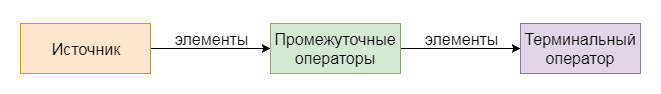
**6.**

**Stream API** — это новый способ работать со структурами данных в функциональном стиле. Stream (поток) API (описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой) — это по своей сути поток данных. Сам термин "поток" довольно размыт в программировании в целом и в Java в частности.

С появлением Java 8 Stream API позволило программистам писать существенно короче то, что раньше занимало много строк кода, а именно — упростить работу с наборами данных, в частности, упростить операции фильтрации, сортировки и другие манипуляции с данными. Если у вас промежуточных операций нет, часто можно и нужно обойтись без стрима, иначе код будет сложнее чем без потока.

## Stream — это объект для универсальной работы с данными. Мы указываем, какие операции хотим провести, при этом не заботясь о деталях реализации. Например, взять элементы из списка сотрудников, выбрать тех, кто младше 40 лет, отсортировать по фамилии и поместить в новый список. Или чуть сложнее, прочитать все json-файлы, находящиеся в папке books, десериализировать в список объектов книг, обработать элементы всех этих списков, а затем сгруппировать книги по автору.

Данные могут быть получены из источников, коими являются коллекции или методы, поставляющие данные. Например, список файлов, массив строк, метод range() для числовых промежутков и т.д. То есть, стрим использует существующие коллекции для получения новых элементов, это ни в коем случае не новая структура данных.

К данным затем применяются операторы. Например, взять лишь некоторые элементы (filter), преобразовать каждый элемент (map), посчитать сумму элементов или объединить всё в один объект (reduce).  


Операторы можно разделить на две группы:  
     - Промежуточные (intermediate) — обрабатывают поступающие элементы и возвращают стрим. Промежуточных операторов в цепочке обработки элементов может быть много.  
     - Терминальные (terminal) — обрабатывают элементы и завершают работу стрима, так что терминальный оператор в цепочке может быть только один.

**7.**

**JAR** (архив Java) — это формат файла пакета, обычно используемый для объединения множества файлов классов Java и связанных метаданных и ресурсов (текста, изображений и т. Д.) В один файл для распространения прикладного программного обеспечения или библиотек на платформе Java.   
Проще говоря, файл JAR — это файл, который содержит сжатую версию файлов .class, аудиофайлов, файлов изображений или каталогов. Мы можем представить файлы .jar в виде zip-файла (.zip), созданного с помощью программного обеспечения WinZip. Даже программное обеспечение WinZip может использоваться для извлечения содержимого .jar. Таким образом, вы можете использовать их для таких задач, как сжатие данных без потерь, архивирование, распаковка и распаковка архива.

Давайте посмотрим, как создать файл .jar и связанные с ним команды, которые помогут нам работать с файлами .jar.

1. Создание файла JAR. Чтобы создать файл .jar, мы можем использовать команду jar cf следующим образом:

jar cf jarfilename inputfiles

Здесь cf обозначает создание файла. Например, предполагая, что наш пакет доступен в каталоге C: /, чтобы преобразовать его в файл jar в файл pack.jar, мы можем дать команду как:

C:\> jar cf pack.jar pack

Теперь файл pack.jar создан

1. Просмотр файла JAR: Чтобы просмотреть содержимое файлов .jar, мы можем использовать команду как:

jar tf jarfilename

Здесь tf представляет табличное представление содержимого файла. Например, чтобы просмотреть содержимое нашего файла pack.jar, мы можем дать команду:

C:/> jar tf pack.jar

Теперь содержимое pack.jar отображается как:

META-INF/

META-INF/MANIFEST.MF

pack/

pack/class1.class

pack/class2.class

..

..

где class1, class2 и т. д. — это классы в пакете. Первые две записи представляют, что существует файл манифеста, созданный и добавленный в pack.jar. Третья запись представляет подкаталог с пакетом имен, а последние два представляют имя файла в пакете каталогов.

Когда мы создаем файлы .jar, он автоматически получает файл манифеста по умолчанию. В архиве может быть только один файл манифеста, и он всегда имеет путь.

META-INF/MANIFEST.MF

Этот файл манифеста полезен для указания информации о других упакованных файлах.

1. Извлечение файла JAR. Чтобы извлечь файлы из файла .jar, мы можем использовать:

jar xf jarfilename

Здесь xf представляет файлы извлечения из файлов jar. Например, чтобы извлечь содержимое нашего файла pack.jar, мы можем написать:

C:\> jar xf pack.jar

Это создаст следующие каталоги в C: /

META-INF

pack // in this directory , we can see class1.class and class2.class.

1. Обновление файла JAR Инструмент Jar предоставляет опцию 'u', которую вы можете использовать для обновления содержимого существующего файла JAR, изменив его манифест или добавив файлы. Основная команда для добавления файлов имеет следующий формат:

jar uf jar-file input-file(s)

Здесь UF представляет файл обновления JAR. Например, чтобы обновить содержимое нашего файла pack.jar, мы можем написать:

C:\>jar uf pack.jar

1. Запуск файла JAR: для запуска приложения, упакованного в файл JAR (требуется заголовок манифеста основного класса), можно использовать следующую команду:

C:\>java -jar pack.jar