1. Что такое Maven? Для чего создан Maven? Как узнать какую версию

Maven вы используете?

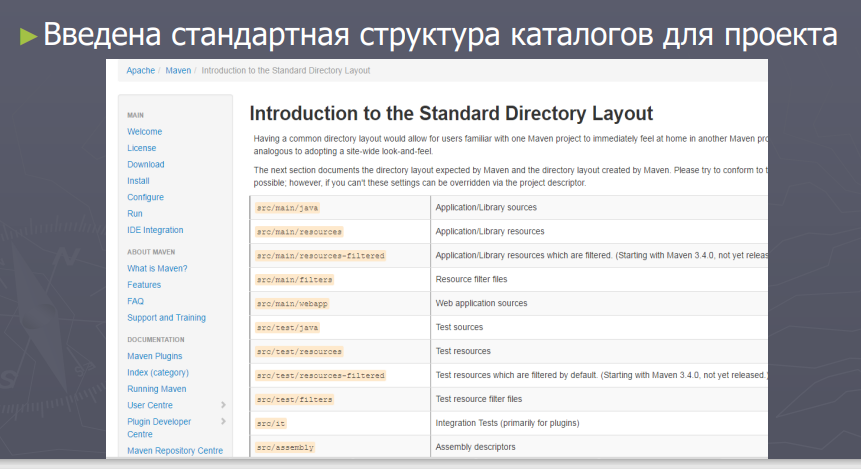
Apache Maven — фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM, являющемся подмножеством XML

* Упрощение процесса сборки
* Обеспечение единообразной системы сборки
* Предоставление качественной информации о проекте
* Поощрение более совершенной практики в области развития

Mvn -version

2. Какая структура каталогов в Maven?

структура каталогов для проекта: корневой каталог – имя проекта (короткое), pom каталог для проектного POM’a, bom – для BOM POM‘а, остальные каталоги по имени модулей.



3. Что такое pom.xml? Какую информацию содержит pom.xml?

pom.xml - это XML-файл, который содержит информацию о конфигурации и деталях проекта, используемых при создании проекта на Maven

4. Что такое супер POM?

Все POM-файлы являются наследниками родительского pom.xml. Этот POM-файл называется Super POM и содержит значения, унаследованные по умолчанию.

5. Какие элементы необходимы для минимального POM?

Обязательные элементы для минимального POM это корневой элемент, modelVersion, GroupID, artifactID и версия.

6. Что такое артефакт? Что является полным именем артефакта?

Артефакт (artefact) - это, по сути, любая библиотека, хранящаяся в репозитарии (месте хранения).

group ID, artifact ID и версию.

7. Что такое зависимости в Maven? Что такое транзитивная зависимость в

Maven?

**Зависимость (dependency)** - это те библиотеки, которые непосредственно используются в вашем проекте для компиляции кода или его тестирования.

**Транзитивная зависимость** - позволяет избегать необходимости изучать и указывать библиотеки, которые требуются для самой зависимости, и включает их автоматически.

8. Что такое плагин в Maven?

**Плагин (plugin)** - это зависимости Maven'а, расширяющие его функционал.

9. Что такое задача в Maven?

**Задача (goal)** - это специальная задача, которая относится к сборке проекта и его управлению. Она может привязываться как к нескольким фазам, так и ни к одной. Задача, которая не привязана ни к одной фазе, может быть запущена вне фаз сборки с помощью прямого вызова.

10.Что такое репозиторий в Maven? Какие типы репозитория существуют в

Maven?

**репозитарий (repository)** - глобальное хранилище всех библиотек, доступных для Maven, это место где хранятся артефакты: jar файлы, pom-файлы, javadoc, исходники, плагины.

* **Локальный (local) репозитарий** - это директория, которая хранится на нашем компьютере.
* **Центральный (central) репозитарий** - это репозитарий, который обеспечивается сообществом Maven.
* **Удалённый (remote) репозитарий**. является репозитарием, определённым самим разработчиком. Там могут храниться все необходимые зависимости.

11.Какой порядок поиска зависимостей Maven?

Когда мы выполняем собрку проекта в Maven, автоматически начинается поиск необходимых зависимостей в следующем порядке:

1. Поиск зависимостей в локальном репозитарии. Если зависимости не обнаружены, происходит переход к шагу 2.
2. Поиск зависимостей в центральном репозитарии. Если они не обнаружены и удалённый репозитарий определён, то происходит переход к шагу 4.
3. Если удалённый репозитарий не определён, то процесс сборки прекращается и выводится сообщение об ошибке.
4. Поиск зависимостей на удалённом репозитарии, если они найдены, то происходит их загрузка в локальный репозитарий, если нет - выводится сообщение об ошибке.

12.Назовите основные фазы жизненного цикла сборки Maven?

В Maven есть следующие три стандартных жизненных цикла:

* **Очистка (clean)** - очищает артефакты, созданные до сборки.
* **Сборка (default or build)** - используется для создания приложения.
* **Создание сайта проекта (site)** - генерирует документацию сайта для проекта.

13.Что делает команда mvn site? Что делает команда mvn clean?

создает веб-сайт проекта. эта команда очищает целевую директорию от созданных в процессе сборки файлов.

14.Из каких фаз состоит жизненный цикл сборки Default (Build)?

* **validate** - проверяет корректность метаинформации о проекте, подтверждает, является ли проект корректным и вся ли необходимая информация доступа для завершения процесса сборки.
* **initialize** - инициализирует состояние сборки, например, различные настройки.
* **generate-sources** - включает любой исходный код в фазу компиляции.
* **process-sources** - обрабатывает исходный код (подготавливает). Например, фильтрует определённые значения.
* **generate-resources** - генерирует ресурсы, которые должны быть включены в пакет.
* **process-resources** - копирует и отправляет ресурсы в указанную директори. Это фаза перед упаковкой.
* **compile** - комплирует исходный код проекта.
* **process-classes** - обработка файлов, полученных в результате компляции. Например, оптимизация байт-кода Java классов.
* **generate-test-sources** - генерирует любые тестовые ресурсы, которые должны быть включены в фазу компиляции.
* **process-test-sources** - обрабатывает исходный код тестов. Например, фильтрует значения.
* **test-compile** - компилирует исходный код тестов в указанную директорию тестов.
* **process-test-classes** - обрабатывает файлы, полученные в результате компиляции исходного кода тестов.
* **test** - запускает тесты классов, используя приемлемый фреймворк юнит-тестирования (например, Junit).
* **prepare-package** - выполняет все необходимые операции для подготовки пакета, непосредственно перед упаковкой.
* **package** - преобразует скомпилированный код и пакет в дистрибутивный формат. Такие как JAR, WAR или EAR.
* **pre-integration-test** - выполняет необходимые действия перед выполнением интеграционных тестов.
* **integration-test** - обрабатывает и распаковывает пакет, если необходимо, в среду, где будут выполняться интеграционные тесты.
* **post-integration-test** - выполняет действия, необходимые после выполнения интеграционных тестов. Например, освобождение ресурсов.
* **verify** - выполняет любые проверки для подтверждения того, что пакет пригоден и отвечает критериям качества.
* **install** - переносит пакет в локальный репозитарий, откуда он будет доступен для использования как зависимость в других проектах.
* **deploy** - копирует финальный пакет (архив) в удалённый репозитарий для, того, чтобы сделать его доступным другим разработчикам и проектам.

15.Какие типы плагинов существуют в Maven?

* **Плагины сборки (Build plugins)** - выполняются в процессе сборки и должны быть конфигурированны внутри блока

<build></build>

файла pom.xml.

* **Плагины отчётов (Reporting plugins)** - выполняются в процессе генерирования сайта и должны быть сконфигурированны внутри блока

<reporting></reporting>

16.Когда Maven использует внешние зависимости?

Если необходимые файлы не найдены ни в центральном, ни на удалённом репозитарии, тогда для решения этой проблемы используются внешние зависимости.

17.Какая команда создает новый проект на основе архетипа?

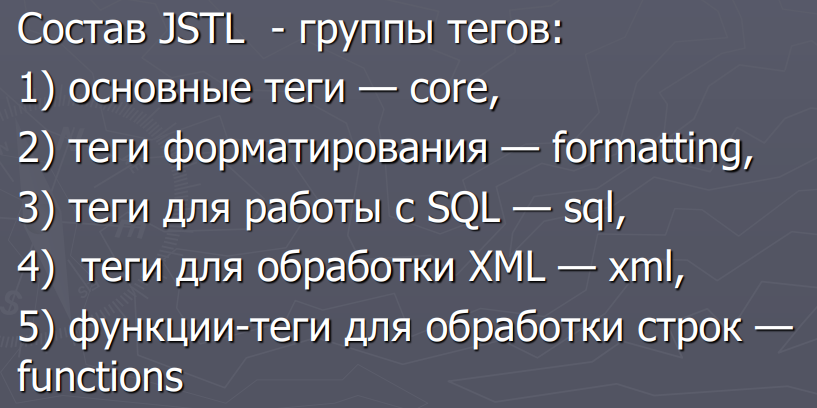
mvn archetype:generate

18.Перечислите теги pom.xml

* **project** - описывает проект, это элемент верхнего уровня во всех файлах pom.xml
* **groupId** - по-сути, это имя пакета. Полностью отражается в структуре каталогов
* **artifactId** - название проекта. В структуре каталогов не отображается
* **version** - версия проекта
* **packaging** - определяет, какой тип файла будет собран. Варианты: pom, jar, war, ear
* **dependencies** - указываются зависимости
* **build** - информация о сборке проекта
* **name** - это уже необязательое описание проекта. В данном случае его название
* **description** - элемент представляет собой общее описание проекта. Это часто используется в генерации документации Maven
* **url** - интернет-страница проекта
* **repositories** - репозитарии для артефактов
* **pluginRepositories** - репозитарии для плагинов Maven

============================12 лаба========================

1. Назовите группы тегов JSTL.



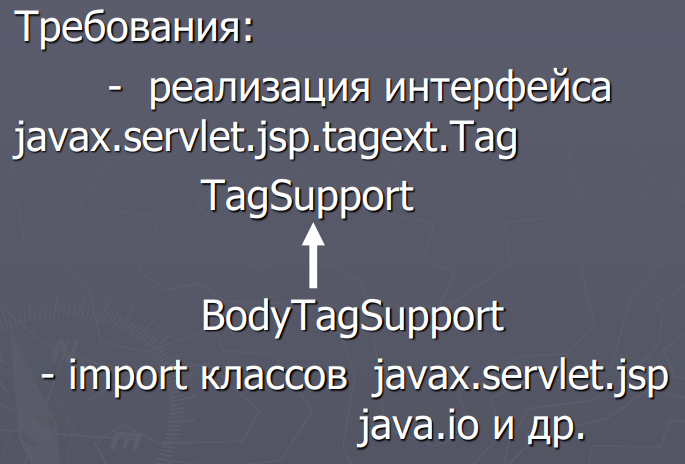
2. Охарактеризуйте группу тегов core.

Основные теги обеспечивают поддержку итерации, условной логики, исключения перехвата, URL-адреса, переадресации или перенаправления ответа и т.д. Чтобы использовать основные теги JSTL, вы должны включить их на страницу JSP, как показано ниже.

* Core: содержит основные теги для наиболее распространенных задач.

Использует префикс "c" и uri "http://java.sun.com/jsp/jstl/core"

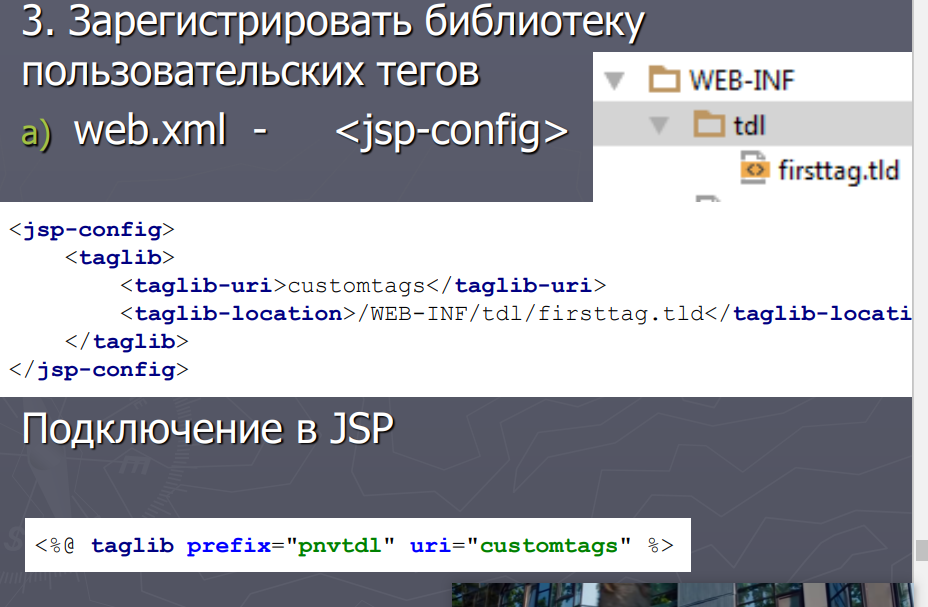
3. Какие требования предъявляются к классу при разработке пользовательского тега?



4. Что такое дескрипторный файла библиотеки тегов \*. Tld

**Дескриптор библиотеки тегов** - это XML-документ, содержащий информацию о библиотеке в целом и о каждом теге, содержащемся в библиотеке. TLD используются веб-контейнером для проверки тегов и средствами разработки страниц JSP.

5. Как зарегистрировать библиотеку пользовательских тегов?



6. Как определить тег с телом?

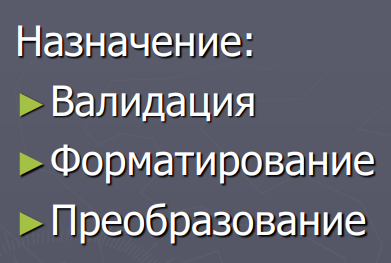
Как и в обычных тегах, между открывающим и закрывающим пользовательскими тегами может находиться тело тега, или **body**. Пользовательские теги могут использовать содержимое элемента **body**–**content**. На данный момент поддерживаются следующие значения для **body–content**:

· **empty** – пустое тело;

· **jsp** – тело состоит из всего того, что может находиться в JSP-файле. Используется для расширения функциональности JSP-страницы;

· **tagdependent** – тело интерпретируется классом, реализующим данный тег. Используется в очень частных случаях.

7. Для чего и как применяют функции-теги?



**=====================13 Лаба===============================**

**1. Для чего используется репозиторий, какие функции он выполняет?**

Репозиторий позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой.

У нас в проекте есть интерфейс репозитори и его наследник абстрактный класс абстракт репозитори, от которого уже наследуются юзер репозитори и персон репозитори

**2. Каково назначение слоя сервисов?**

Сервисный слой, реализующий взаимодействие между слоями представления и бизнес-логики. Примерами реализаций сервисного слоя являются контроллеры, веб-сервисы и слушатели очередей сообщений.

**3. Расскажите что такое паттерн FrontController.**

Один контроллер обрабатывает все запросы к веб-сайту.

В сложных веб-сайтах есть много одинаковых действий, которые надо производить во время обработки запросов. Это, например, контроль безопасности, многоязычность и настройка интерфейса пользователя. Когда поведение входного контроллера разбросано между несколькими объектами, дублируется большое количество кода. Помимо прочего возникают сложности смены поведения в реальном времени.

Паттерн Front Controller объединяет всю обработку запросов, пропуская запросы через единственный объект-обработчик. Этот объект содержит общую логику поведения, которая может быть изменена в реальном времени при помощи декораторов. После обработки запроса контроллер обращается к конкретному объекту для отработки конкретного поведения.

**4. Как используется абстрактная фабрика в вашем проекте?**

**Абстрактная фабрика** — [порождающий шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D0%BE%D0%B6%D0%B4%D0%B0%D1%8E%D1%89%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

Шаблон реализуется созданием абстрактного класса CommandFactory, от которого наслед комманды

**5. Расскажите о взаимодействии jsp-servlet-jsp.**

Жсп визуал, сервлет логика