Создание enterprise приложения

1. Создадим новую базу данных в MySQL и добавим необходимые таблицы.

```
create database news;
use news;

create table news (
  id int not null auto_increment,
  title char(200) not null,
  short_text text not null,
  full_text text not null,
  date datetime not null,
  topic_id int not null,
  primary key (id));

create table  topic (
  id int not null auto_increment,
  name char(100) not null,
  topic_order int not null,
  primary key (id));
```

2. Создадим новый maven-проект. Пусть структура проекта будет выглядеть следующим образом:

news – корневой проект, все остальные проекты добавляются как модули в него

news-ear – ear-архив приложения, при сборке создается архив с расширением .ear

news-web – веб-компонент нашего приложения, при сборке запаковывается внутрь news-ear как .war-архив

news-ejb-api – проект, который содержит интерфейсы для ejb-компонент. Вынести Remote, Local интерфейсы в отдельный проект полезно, т.к. тогда можно избежать зависимости веб-проекта от реализации ejb-компонент.

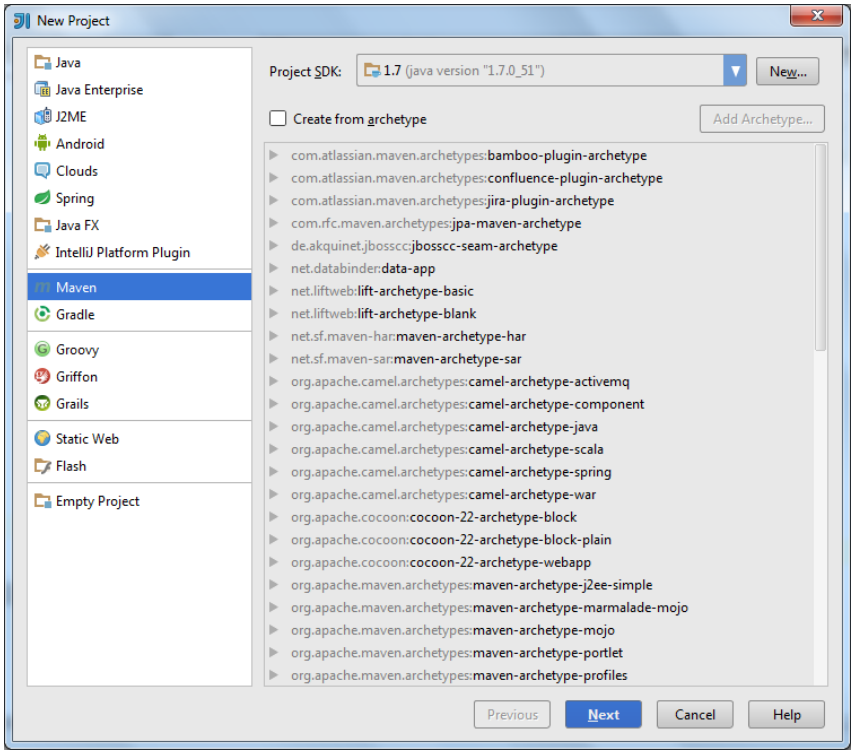
news-ejb – реализация ejb-модуля нашего проекта

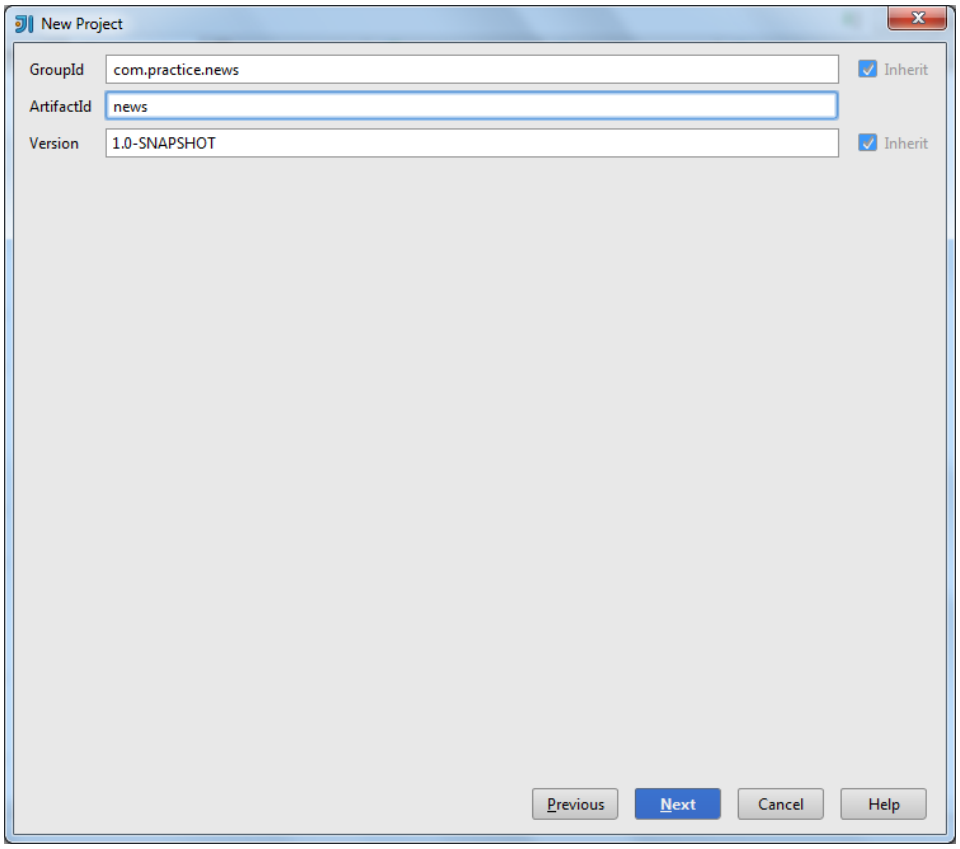
news-entities – набор entity-классов нашего проекта, подключается зависимостью в news-ejb

news-model – классы модели нашего проекта. Содержит Transfer Objects, которые используются для отображения на web. Лучше сделать так, чтобы entity-компоненты не использовались в веб-сервисах и для отображения веб-интерфейса. Поэтому мы сделаем

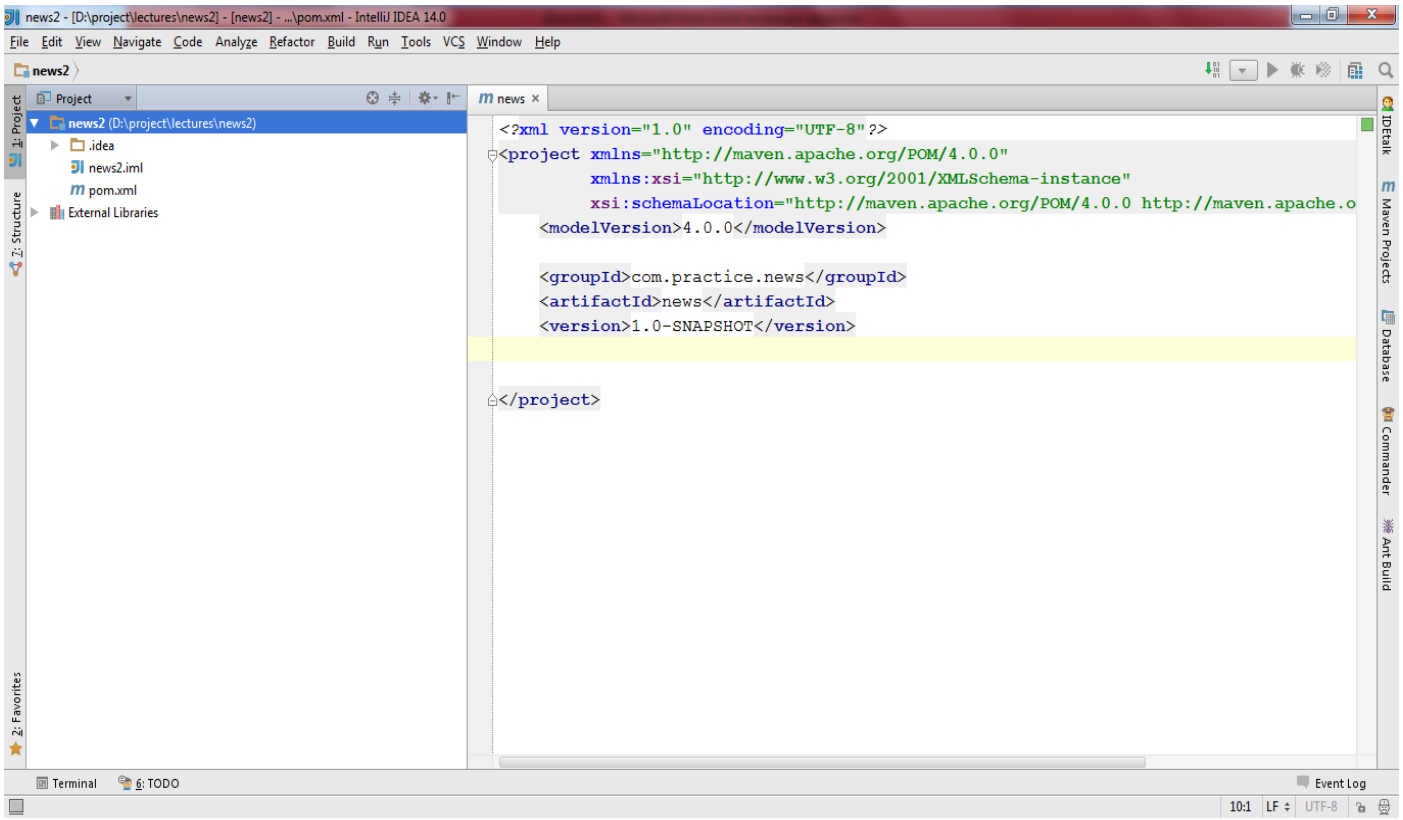
похожую на entity по содержимому иерархию классов.

Создаем новый проект **news**





В создавшемся проекте папку src можно удалить



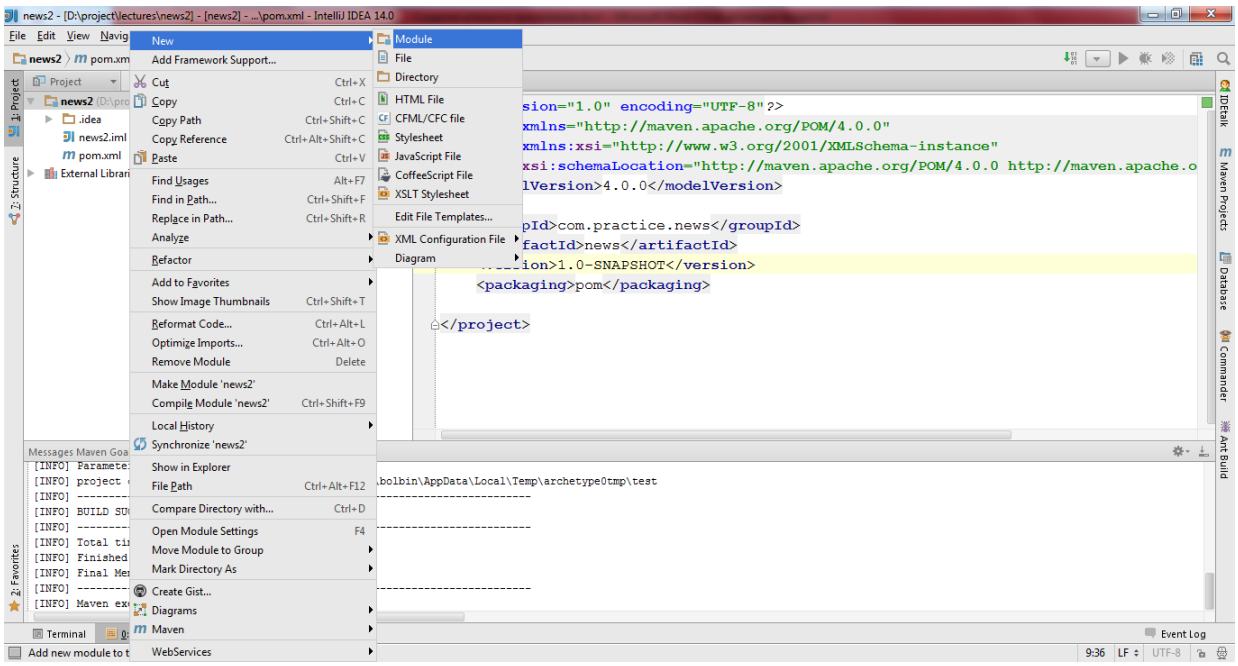
после тэга <version> добавим атрибут packaging pom, чтобы указать тип упаковки проекта
<packaging>pom</packaging>

Сразу можем добавить зависимость Log4j в файл pom.xml . Тогда библиотека будет доступна во всех подпроектах, которые мы создадим ниже

```
<dependencies>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
    <type>jar</type>
  </dependency>
</dependencies>
```

3. Теперь создадим модули проекта

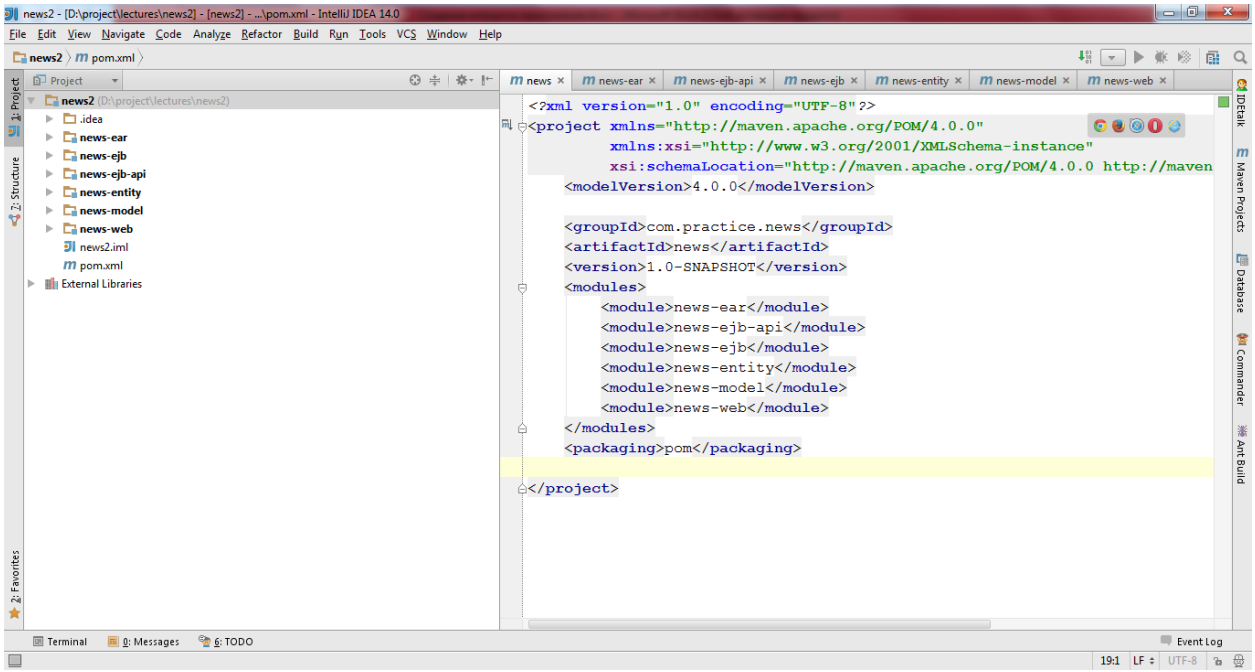
- **news-ear** (правой кнопкой по проекту new->module->maven и укажем артефакт news-ear)



В дереве проекта появляется новый модуль, в главном pom-файле добавляется директива modules

```
<modules>
   <module>news-ear</module>
</modules>
```

Теперь аналогично создадим остальные модули news-ejb-api, news-ejb, news-entity, news-module, news-web



4. Выполним конфигурацию созданных модулей и добавим необходимые зависимости

4.1 **news-ear**

Откроем pom.xml и добавим элемент

<packaging>ear</packaging> (тип собираемого архива)

Добавим зависимости **news-ejb**, **news-web** как war и ejb модули

```
<dependencies>
  <dependency>
    <groupId>com.practice.news</groupId>
    <artifactId>news-ejb</artifactId>
    <version>1.0-SNAPSHOT</version>
    <type>ejb</type>
  </dependency>

  <dependency>
    <groupId>com.practice.news</groupId>
    <artifactId>news-web</artifactId>
    <version>1.0-SNAPSHOT</version>
    <type>war</type>
  </dependency>
</dependencies>
```

Подключим maven-ear и maven-compiler плагины для корректной сборки ear-проекта

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-ear-plugin</artifactId>
      <version>2.6</version>
      <configuration>
        <version>6</version>
        <defaultLibBundleDir>lib</defaultLibBundleDir>
      </configuration>
    </plugin>
  </plugins>
</build>
```

4.2 **news-ejb**

Укажем в pom.xml проекта packaging-тип ejb и зависимости от проектов news-model, news-ejb-api, news-entites, а также необходимые нам внешние зависимости для библиотек mysql-connector-java, hibernate-jpa-2.1-api, hibernate-entitymanager, javaee-api

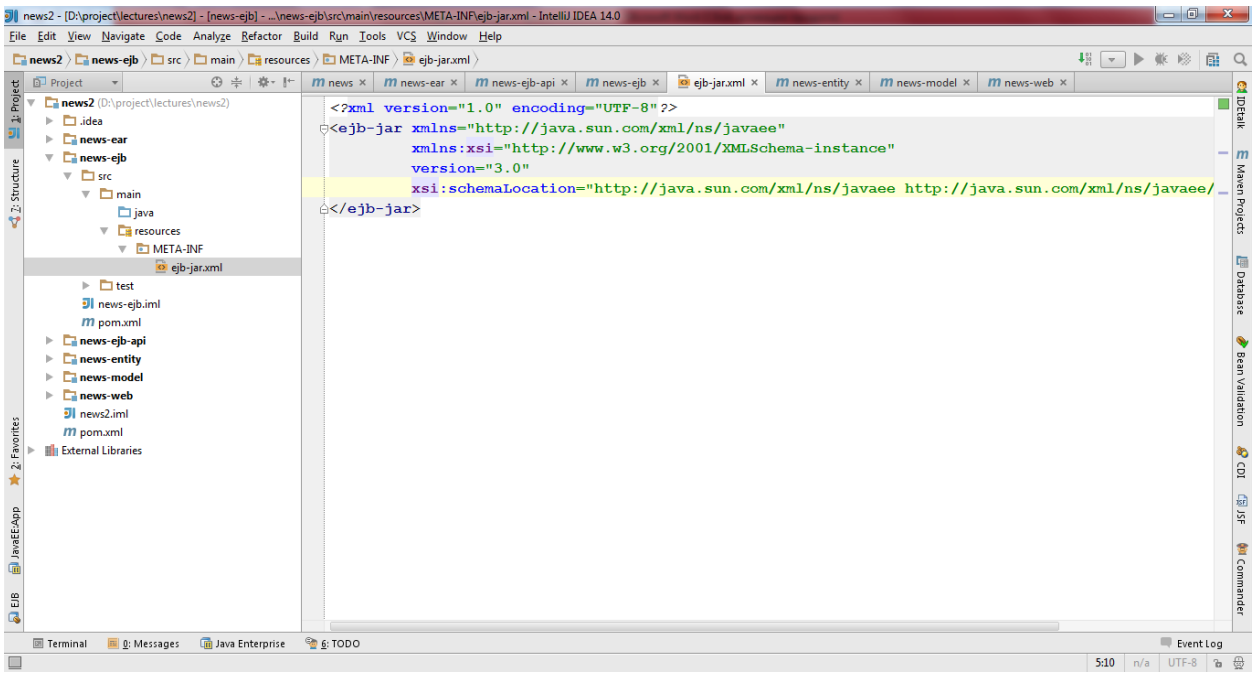
В результате содержимое pom.xml для проекта news-ejb:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>news</artifactId>
    <groupId>com.practice.news</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>

  <modelVersion>4.0.0</modelVersion>
  <artifactId>news-ejb</artifactId>
```

```
  <packaging>ejb</packaging>

  <dependencies>
    <dependency>
      <artifactId>news-ejb-api</artifactId>
      <groupId>com.practice.news</groupId>
      <version>1.0-SNAPSHOT</version>
    </dependency>

    <dependency>
      <artifactId>news-entity</artifactId>
      <groupId>com.practice.news</groupId>
      <version>1.0-SNAPSHOT</version>
    </dependency>

    <dependency>
      <artifactId>news-model</artifactId>
      <groupId>com.practice.news</groupId>
      <version>1.0-SNAPSHOT</version>
    </dependency>

    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.25</version>
    </dependency>

    <dependency>
      <groupId>org.hibernate.javax.persistence</groupId>
      <artifactId>hibernate-jpa-2.1-api</artifactId>
      <version>1.0.0.Final</version>
    </dependency>

    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-entitymanager</artifactId>
      <version>4.3.0.Final</version>
    </dependency>

    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>6.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

Теперь добавим дескриптор развертывания для ejb-модуля. Регистрация бинов необязательна для javaee5 и выше, поэтому создадим пустой файл ejb-jar.xml
Создаем в папке ресурсов проекта **news-ejb** директорию META-INF. Внутри нее создаем файл ejb-jar.xml. Содержимое файла

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      version="3.0" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd">
</ejb-jar>
```



4.3. **news-entity**

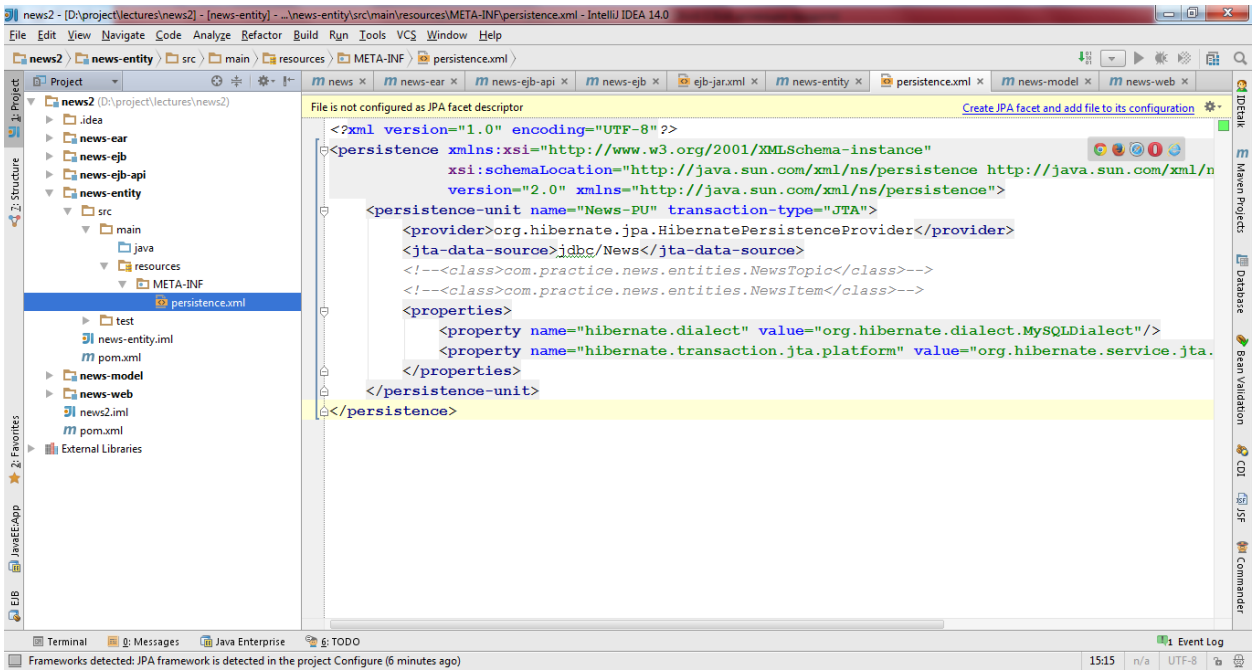
Здесь мы предполагаем создать entity-классы нашего приложения, еще их называют доменные сущности. Поэтому помимо pom.xml необходимо создать persistence-unit. Сначала сконфигурируем pom-файл. Здесь нам также понадобятся зависимости для jpa и hibernate

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>news</artifactId>
        <groupId>com.practice.news</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>news-entity</artifactId>
    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>6.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.25</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate.javax.persistence</groupId>
            <artifactId>hibernate-jpa-2.1-api</artifactId>
            <version>1.0.0.Final</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>4.3.0.Final</version>
        </dependency>
    </dependencies>
</project>
```

Теперь создадим директорию META-INF в папке resources проекта news-entity и в этой папке создадим файл **persistence.xml** для конфигурирования маппинга базы данных
В созданном файле нам необходимо указать
- имя persistence-unit (допустим, News-PU)
- тип транзакционной модели (в случае, когда создается enterprise-приложение, **JTA**)
- data-source (имя jdbc-ресурса для связи с базой данных, пока мы укажем имя **jdbc/News,** далее, когда будем настраивать сервер приложений, создадим ресурс с этим именем)
- настройки специфичные для hibernate при работе с mysql
Entity-классы пока регистрировать не будем

Содержимое файла

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
        version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="News-PU" transaction-type="JTA">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <jta-data-source>jdbc/News</jta-data-source>

    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
      <property name="hibernate.transaction.jta.platform" value="org.hibernate.service.jta.platform.internal.SunOneJtaPlatform" />
    </properties>
  </persistence-unit>
</persistence>
```



4.4. **news-web**

Web-модуль проекта требует специфичной конфигурации.

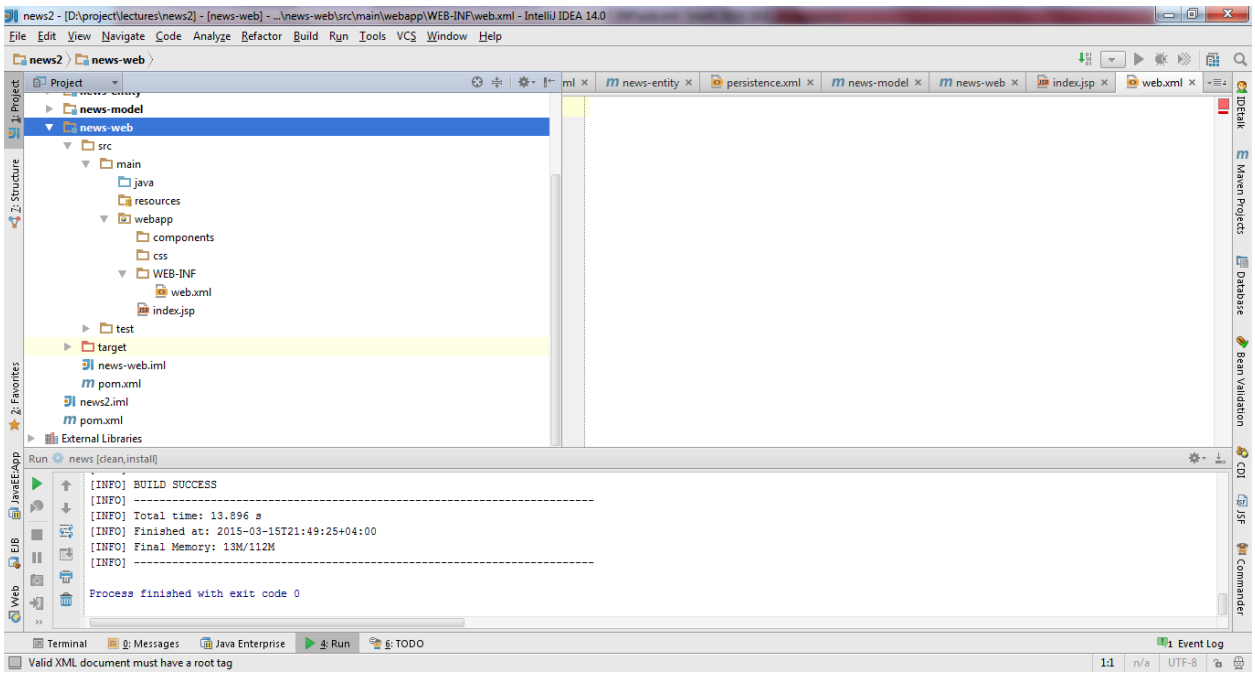
- сконфигурируем pom.xml файл. Нужно указать тип packaging = war, добавить зависимости для javaee-webapi, servlet-api, jstl, а также для проектов news-model, news-ejb-api, полезной будет нам библиотека apache-commons. Наконец, нужно подключить maven-плагин для сборки war-проектов

Содержимое pom-файла:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <parent>
    <artifactId>news</artifactId>
    <groupId>com.practice.news</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>

  <modelVersion>4.0.0</modelVersion>
  <artifactId>news-web</artifactId>
  <packaging>war</packaging>

  <properties>
    <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <!-- dependency for web-api-->
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-web-api</artifactId>
      <version>6.0</version>
      <scope>provided</scope>
    </dependency>

    <!-- dependency for servlet-api-->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
    </dependency>

    <!--dependency for ejb usage (lockup or inject) -->
    <dependency>
      <groupId>javax.ejb</groupId>
      <artifactId>javax.ejb-api</artifactId>
      <version>3.2</version>
    </dependency>

    <!--dependency for JSTL-->
    <dependency>
      <groupId>jstl</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>

    <!-- Apache Commons additional language tools-->
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
      <version>3.1</version>
    </dependency>

    <!-- project modules-->
    <dependency>
      <groupId>com.practice.news</groupId>
      <artifactId>news-ejb-api</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>

    <dependency>
      <groupId>com.practice.news</groupId>
      <artifactId>news-model</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
```

```
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
          <compilerArguments>
            <endorseddirs>${endorsed.dir}</endorseddirs>
          </compilerArguments>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.1.1</version>
        <configuration>
          <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <version>2.1</version>
        <executions>
          <execution>
            <phase>validate</phase>
            <goals>
              <goal>copy</goal>
            </goals>
            <configuration>
              <outputDirectory>${endorsed.dir}</outputDirectory>
              <silent>true</silent>
              <artifactItems>
                <artifactItem>
                  <groupId>javax</groupId>
                  <artifactId>javaee-endorsed-api</artifactId>
                  <version>6.0</version>
                  <type>jar</type>
                </artifactItem>
              </artifactItems>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

- Теперь создадим в папке src/main папку webapp. Внутри webapp создадим служебную директорию WEBINF, а также папки для стилей и jsp-страничек (css, components) . в корне webapp создадим файл **news.jsp**. В директории WEBINF создадим файл web.xml



Добавим содержимое в файл-дескриптор развертывания web.xml. Укажем, welcome-file index.jsp и отображаемое имя приложения

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <display-name>News Web Application</display-name>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>


</web-app>
```

4.5 В проект **news-ejb-api** добавим зависимость от проекта **news-model, java-ee-api**

```
<dependencies>
  <dependency>
    <artifactId>news-model</artifactId>
    <groupId>com.practice.news</groupId>
    <version>1.0-SNAPSHOT</version>
  </dependency>

  <dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-api</artifactId>
    <version>6.0</version>
    <scope>provided</scope>
  </dependency>

</dependencies>
```

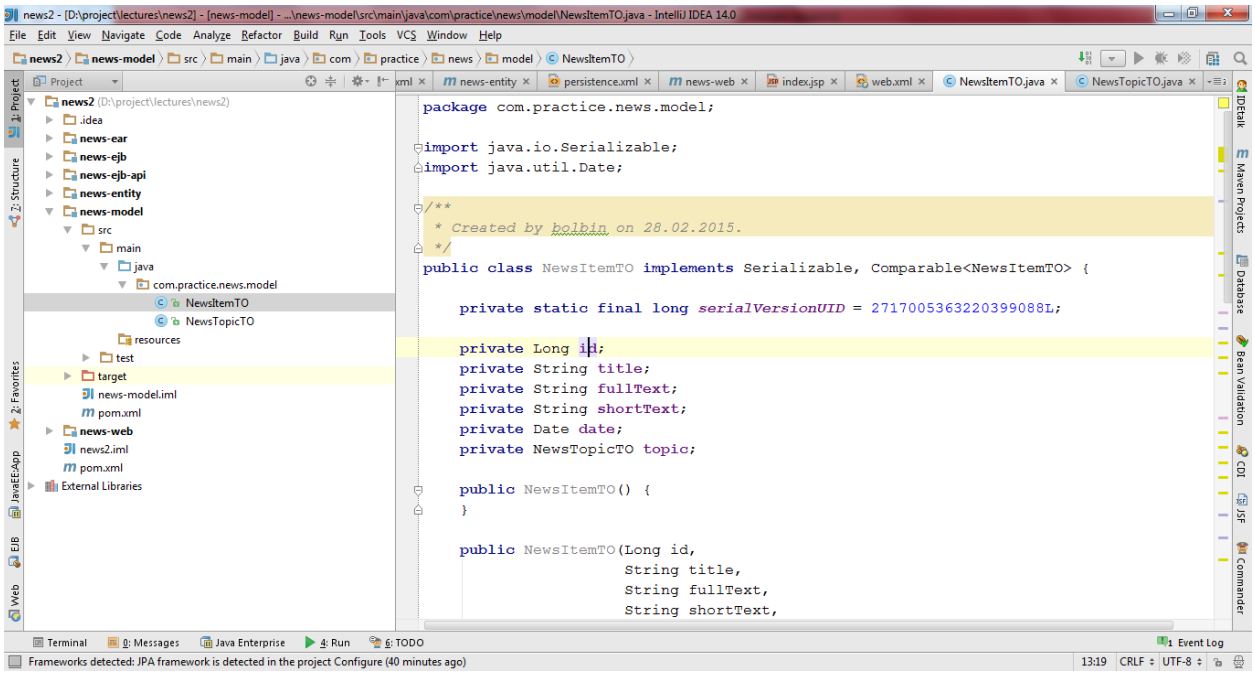
5. Создадим объекты модели приложения. В проекте news-model создадим пакет **com.practice.news.model**, внутри которого создадим классы модели

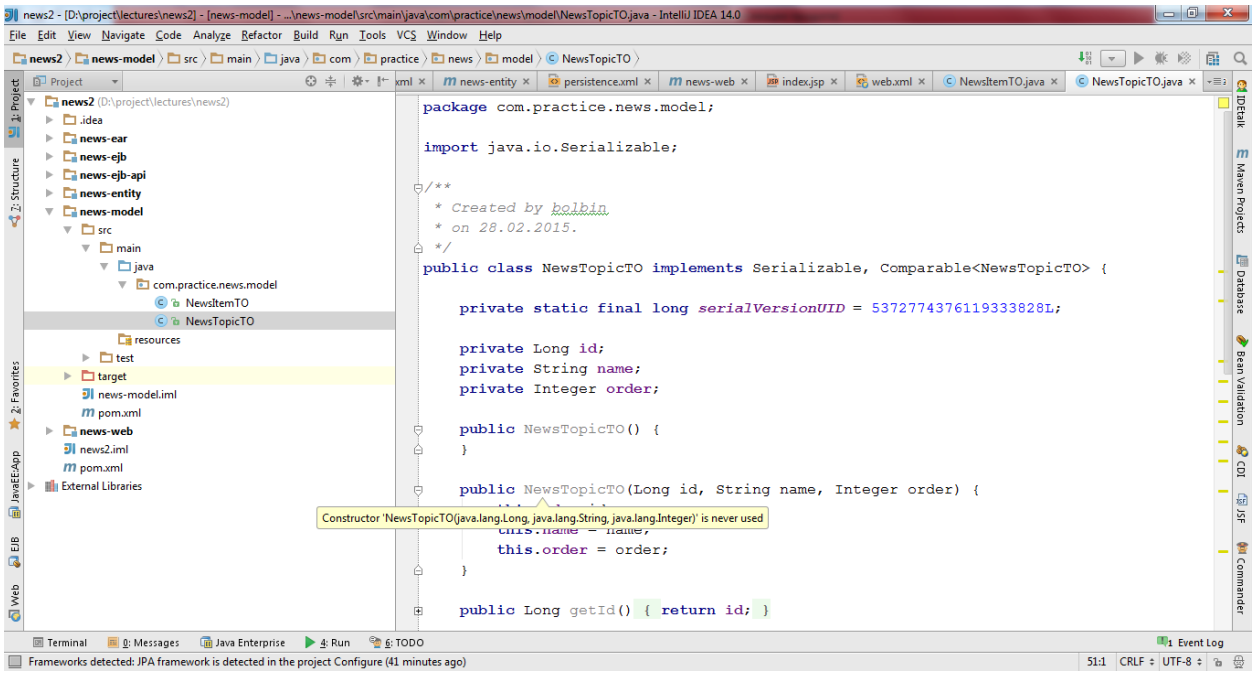
NewsItemTO(id:Long, title:String, fullText:String, shortText:String, date:Date, topic:NewsTopicTO)

и

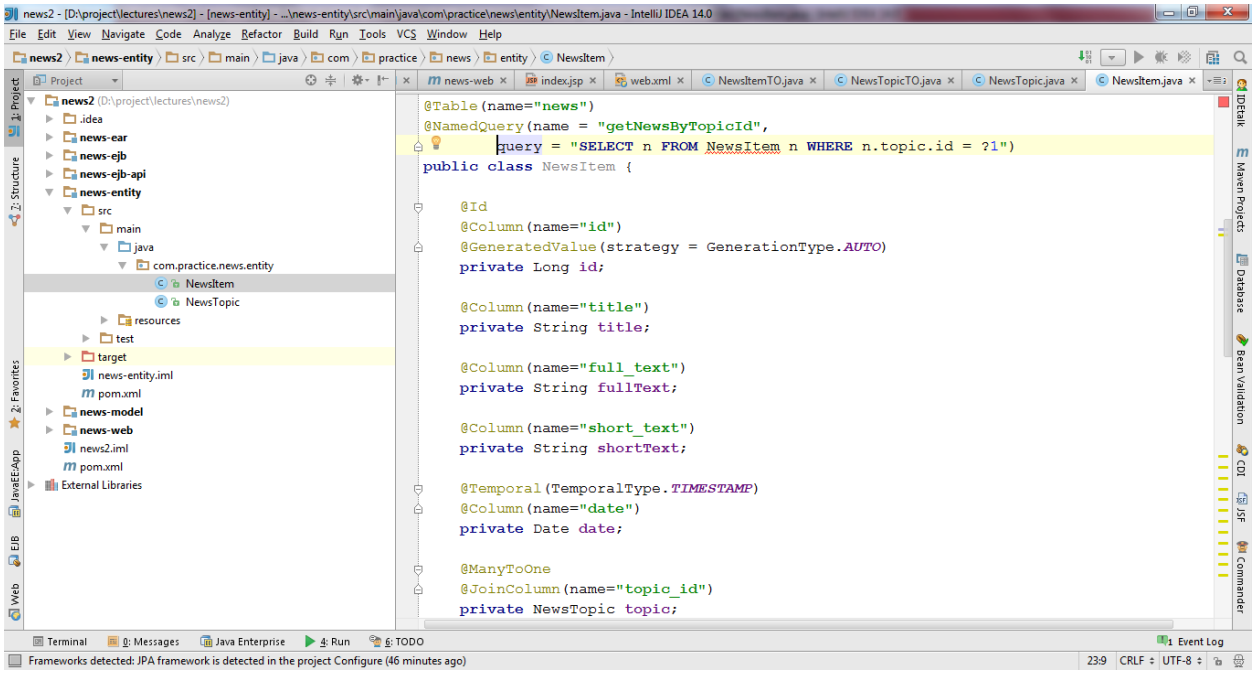
NewsTopicTO(id:Long, name:String, order:Integer)

Сгенерируем конструкторы, конструкторы по умолчанию, набор гет и сет методов. Классы должны реализовывать интерфейс Serializable. Для сортировки новостей и групп при отображении реализуем интерфейс Comparable.





6. Создадим пакет и entity-классы NewsItem, NewsTopic в модуле news-entity (см. код приложения).



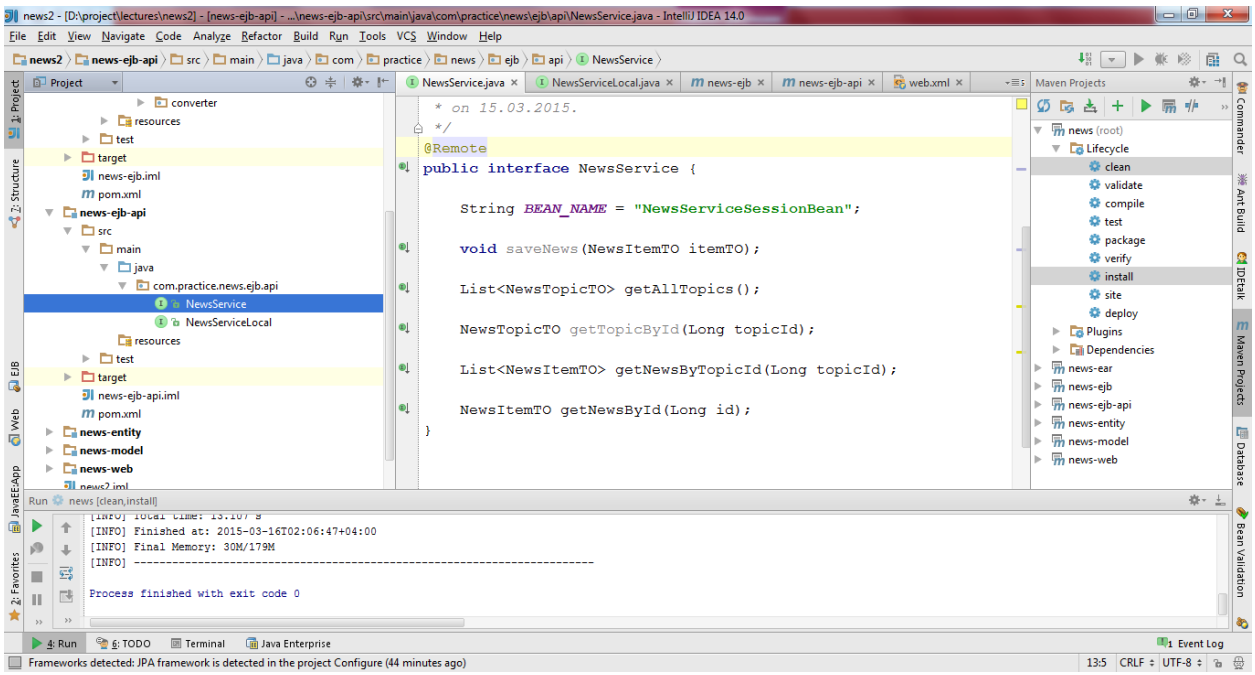
7. В файле persistence.xml регистрируем entity-классы (опционально)

<class>com.practice.news.entity.NewsTopic</class>

<class>com.practice.news.entity.NewsItem</class>

8. Создадим интерфейсы для ejb-компонент

В проекте **news-ejb-api** создадим пакет **com.practice.news.ejb.api**
В нем создадим интерфейсы NewsService, NewsServiceLocal – remote и local интерфейсы будущего ejb-компонента. В них создадим методы для логики работы с новостями (получение, по категориям, по id, получение категорий, сохранение и т.п.)




```
@Remote
public interface NewsService  {

   String BEAN_NAME = "NewsServiceSessionBean";

   void saveNews(NewsItemTO itemTO);

   List<NewsTopicTO> getAllTopics();

   NewsTopicTO getTopicById(Long topicId);

   List<NewsItemTO> getNewsByTopicId(Long topicId);

   NewsItemTO getNewsById(Long id);
}
```

```
@Local
public interface NewsServiceLocal extends NewsService {
}
```

9. В проекте news-ejb создадим пакеты **com.practice.news.ejb.bean, com.practice.news.ejb.converter**

В пакете converter реализуем классы для создания сущностей entity из TO и наоборот (см. NewsItemConverter, NewsTopicConverter). Этими классами мы будем пользоваться в реализации ejb-компонента

10. В пакете **com.practice.news.ejb.bean** создаем класс NewsServiceSessionBean (реализацию EJB). С помощью соответствующих аннотаций указываем remote и local интерфейсы бина, а также указываем, что он является Stateless.

```
@Remote(NewsService.class)
@Local(NewsServiceLocal.class)
@Stateless(mappedName = NewsService.BEAN_NAME)
```

```
public class NewsServiceSessionBean implements NewsService, NewsServiceLocal {
}
```

Реализуем методы интерфейсов, с помощью аннотаций подключаем persistence unit:

```
@PersistenceContext(unitName = "News-PU")
private EntityManager entityManager;
```

Реализуем логику работы компонента. Обращения к базе данных осуществляем через entity-manager (см. реализацию класса)

11. Реализация веб-компонента. Создадим в src/main/java-папке проекта news-web пакет **com.practice.news.servlet**
В нем создаем класс **NewsServlet,** который наследует HttpServlet
Для того, чтобы обращаться к ejb-компоненту, создаем поле и используем аннотацию @EJB:

```
@EJB(mappedName = NewsService.BEAN_NAME)
private NewsServiceLocal newsEJB;
```

При этом в качестве типа ejb-компонента указываем интерфейс
Переопределяем метод doGet(), в нем реализуем логику получения новостей по id темы, либо детали новости, если был передан ее идентификатор как get-параметр http-запроса news=.

Если параметр news был передан в http-запросе, то получаем содержимое новости по ее идентификатору, в request-атрибут “news” кладем полученный объект newsItemTO и совершаем редирект на страницу отображения news.jsp .

Если параметр news запроса не был передан, то ищем все новости по id текущей темы (параметр запроса “topic”) , получаем список новостей, кладем его в атрибут response “newsList”. Если в запросе не был передан параметр id темы, то читаем новости по первой теме.

В обоих случаях для вывода меню получаем список всех тем и кладем их в атрибут request.setAttribute(“topics”, allTopics).

Теперь нужно зарегистрировать сервлет в конфигурационном файле web.xml.

Указываем servlet и маппинг в дескрипторе web.xml:

```
<servlet>
  <servlet-name>NewsServlet</servlet-name>
  <servlet-class>com.practice.news.servlet.NewsServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>NewsServlet</servlet-name>
  <url-pattern>/index.jsp</url-pattern>
</servlet-mapping>
```

Таким образом, наш сервлет будет срабатывать при каждом обращении к стартовой странице index.jsp.

12. Создадим в папке css файл стилей main.css

13. В папке components создадим файлы:

topics.jsp – для отображения меню со списком тем, его содержимое

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:forEach var="topic" items="${requestScope.topics}">
  <a href="${pageContext.request.contextPath}/?topic=${topic.id}">${topic.name}</a><br>
</c:forEach>
```

т.е. циклично итерируемся по списку тем, которые лежат в параметре request “topics” (в jsp для этого нужно выполнить обращение requestScope.<имя_параметра>), и выводим ссылки на соответствующий пункт меню. Обратите внимание, что вызов topic.getName() метод класса NewsTopicTO преобразуется в **topic.name** (т.е. «get» опускается, первая заглавная буква преобразуется в нижний регистр). К сет-методам из jstl обращаться нельзя.

news_list.jsp – для отображения списка новостей. Аналогично, с помощью forEach итерируемся по списку новостей из параметров request, выводим содержимое новостей и ссылку на получение детального представления.

news_details.jsp - детальное содержимое новости

Файл **news.jsp** - основная страница отображения ленты новостей. Его содержимое:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <link rel="stylesheet" type="text/css" href="css/main.css"/>
  <title>News List</title>
</head>
<body>
<table width="100%">
  <tr valign="top">
    <td width="150px" class="topics">
      <jsp:include page="components/topics.jsp"/>
    </td>
    <td>

      <c:if test="${! empty requestScope.error}">
        <div class="error">${requestScope.error}</div>
      </c:if>

      <c:choose>
        <c:when test="${! empty requestScope.news}">
          <jsp:include page="components/news_details.jsp"/>
        </c:when>
        <c:otherwise>
          <jsp:include page="components/news_list.jsp" />
```

```
        </c:otherwise>
      </c:choose>
    </td>
  </tr>
</table>
</body>
</html>
```

Как видим, файл содержит подключения страничек, которые мы описали выше с помощью тэга jsp:include. Причем, выбор между детальным представлением и списком

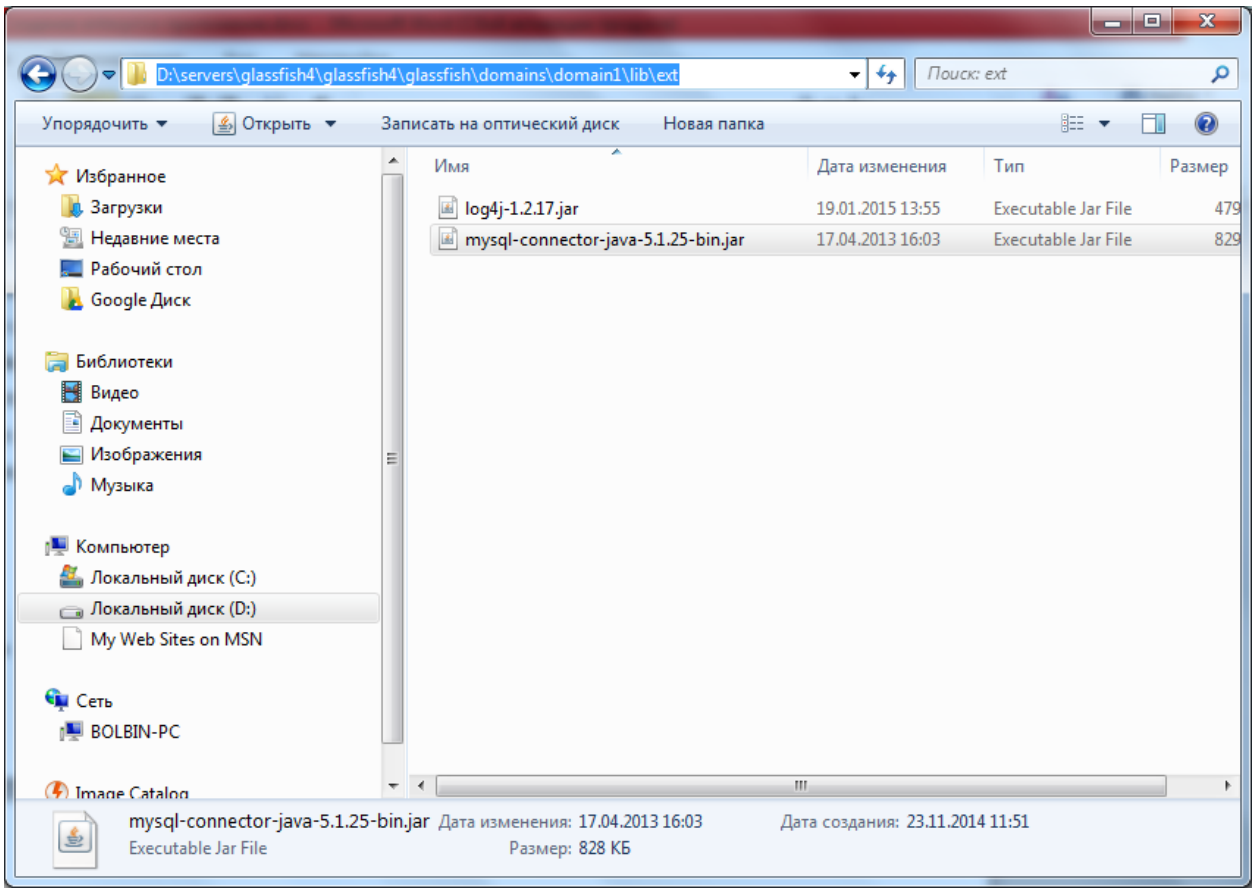
```
<c:when test="${! empty requestScope.news}">
  <jsp:include page="components/news_details.jsp"/>
</c:when>
<c:otherwise>
  <jsp:include page="components/news_list.jsp" />
</c:otherwise>
```

основан на наличии в атрибутах http-request параметра news (детальное представление новости).

Таким образом, когда пользователь открывает index.jsp страничку приложения выполняется примерная последовательность действий (с точки зрения функциональности и взаимодействия компонент)

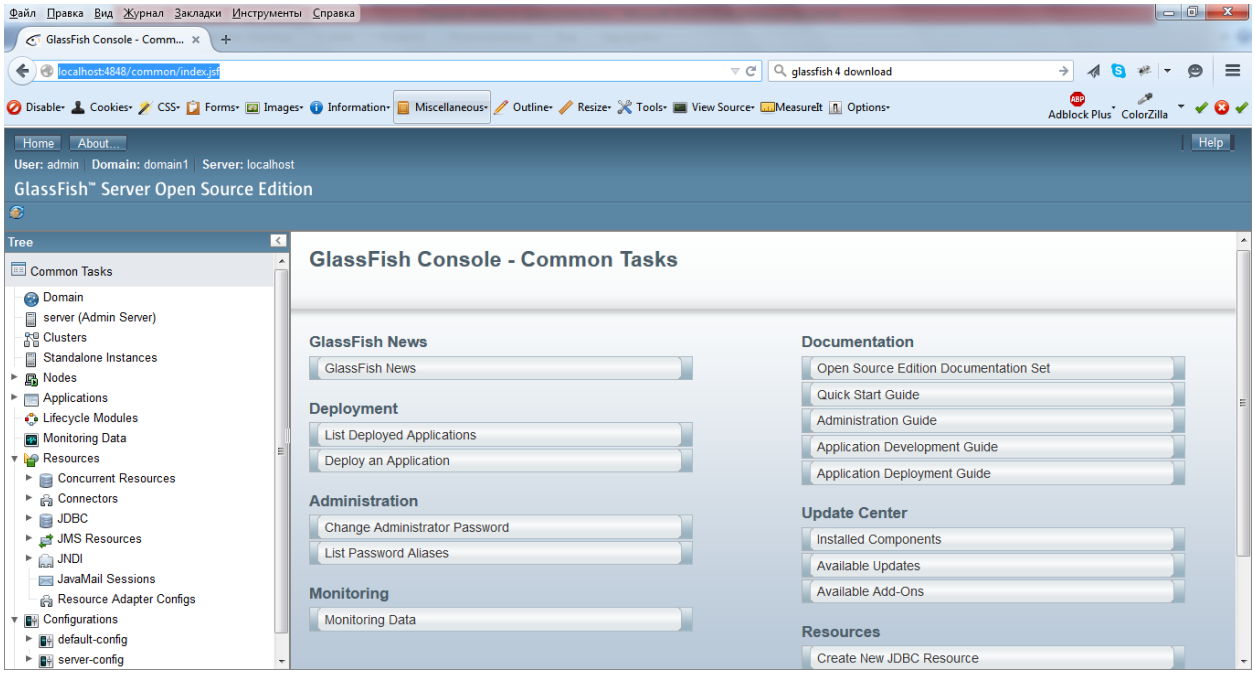
- Web-контейнер по файлу web.xml находит NewsServlet, маппинг которого соответствует данному url и передает управление ему
- Сервлет запрашивает ejb-компонент и инициализирует переменную newsEJB
- Сервлет выполняет метод doGet, обрабатывает параметры запроса и выполняет обращение к ejb для того, чтобы получить список новостей
- EJB-контейнер передает управление конкретному ejb-объекту
- ejb-объект выполняет метод getNewsByTopicId, при этом осуществляя транзакционное обращение к базе данных через jdbc и hibernate/jpa, границы транзакции при этом ограничены началом и концом выполнения метода
- ejb-объект через ejb-контейнер возвращает список полученных объектов сервлету
- сервлет устанавливает атрибут http-запроса и переадресует запрос jsp-странице news.jsp
- news.jsp выполняет логику представления данных, которые были переданы в httpRequest.

14. Осуществляем сборку проекта с помощью средств IDEA или из консоли с помощью команды mvn –e clean install (из корня директории всего проекта)
15. Устанавливаем сервер приложений glassfish (https://glassfish.java.net/download.html), скачиваем и распаковываем архив. После этого в установленной директории находим директорию, где лежат библиотеки домена (glassfish/domains/domain1/lib) и в ext кладем mysql-jdbc-connector



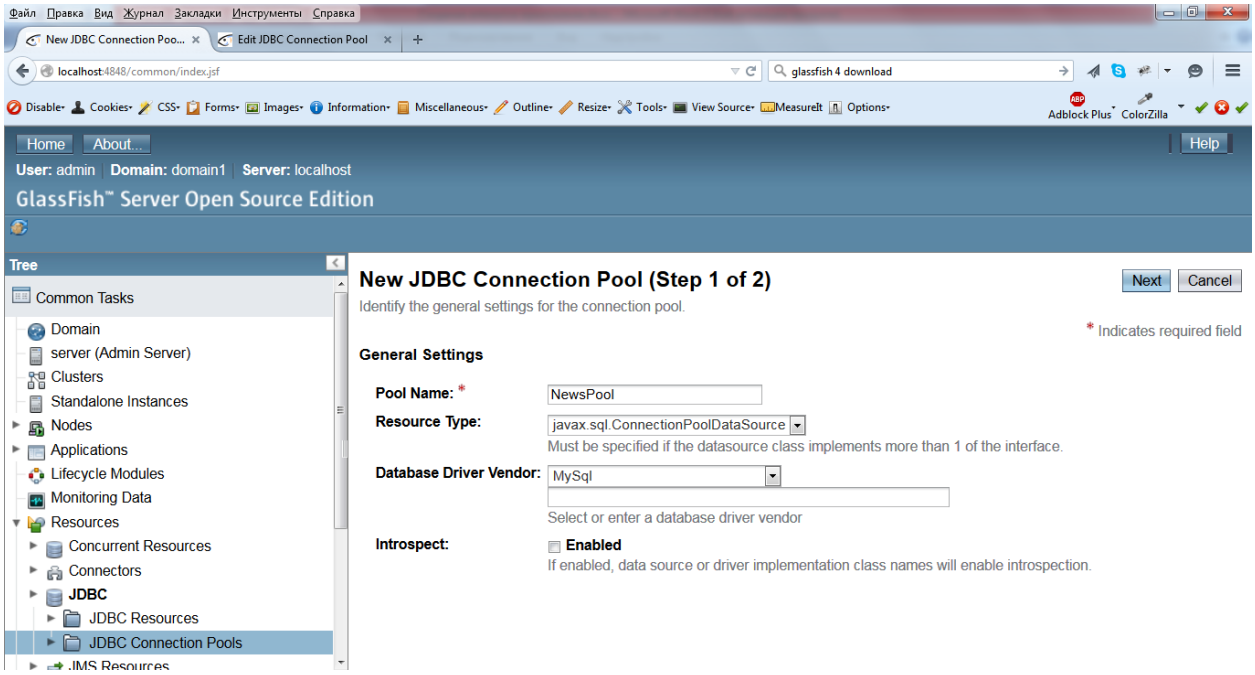
16. Выполняем старт домена glassfish. Для этого переходим в директорию, куда был установлен glassfish, в папке bin запускаем утилиту **asadmin.bat** и выполняем команду **start-domain domain1**

17. В браузере открываем ссылку на веб-панель localhost:4848

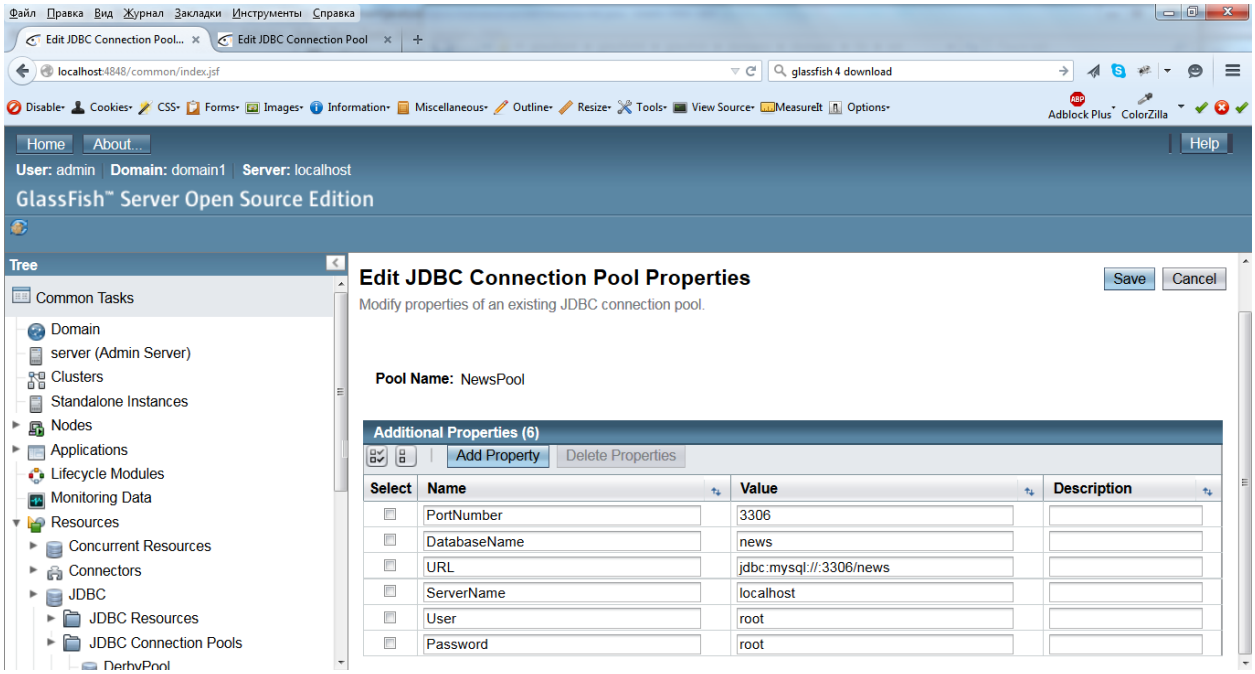


И переходим в раздел jdbc/jdbc connection pools.

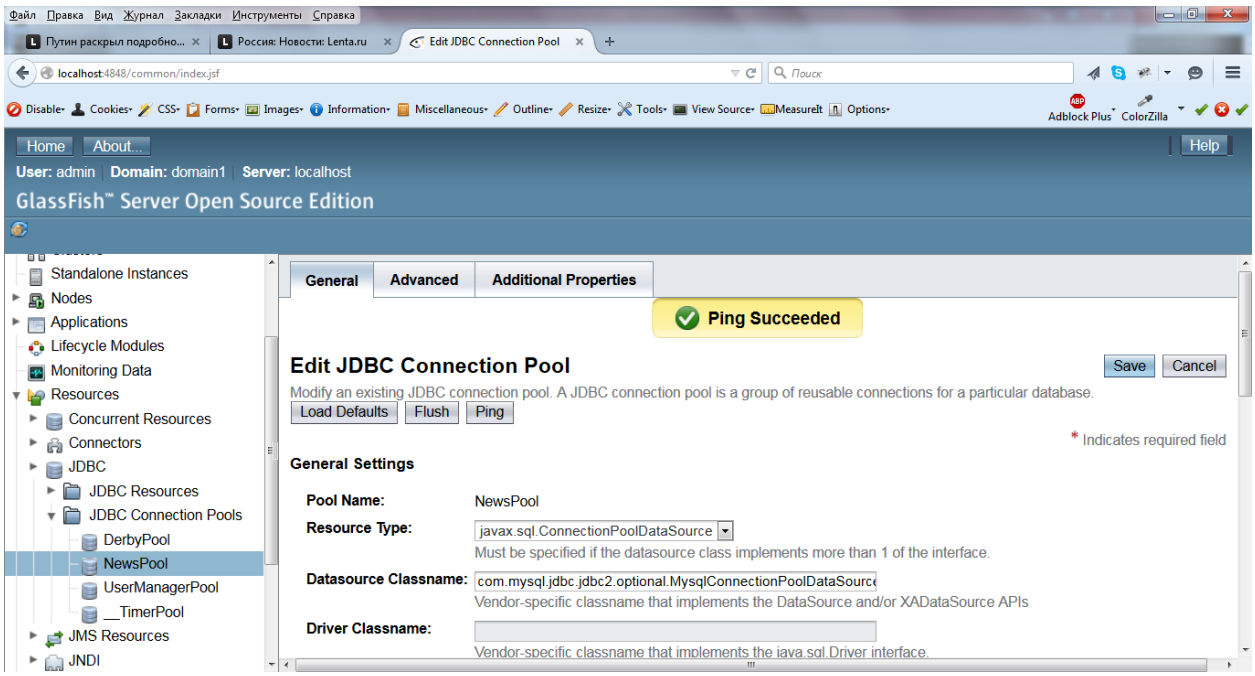
Здесь нам нужно создать новый пул соединений для нашей базы данных.
Нажимаем new и указываем параметры для создания mysql connection pool



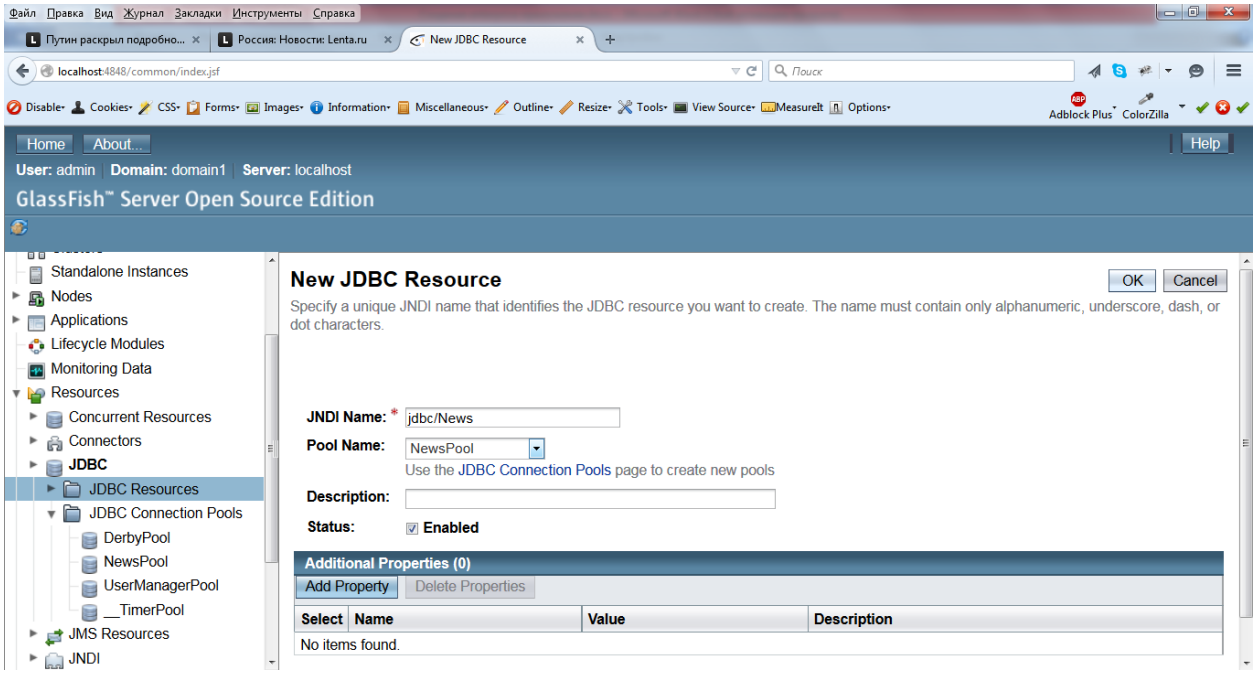
В нижней части страницы удаляем дефолтные настройки (properties) и создаем свои параметры для соединения с базой данных



User/Passsword соответствует логину и паролю пользователя базы данных.
Нажимаем Save. Можно выбрать в списке пулов наш NewsPool и нажать кнопку Ping, чтобы проверить работоспособность соединения



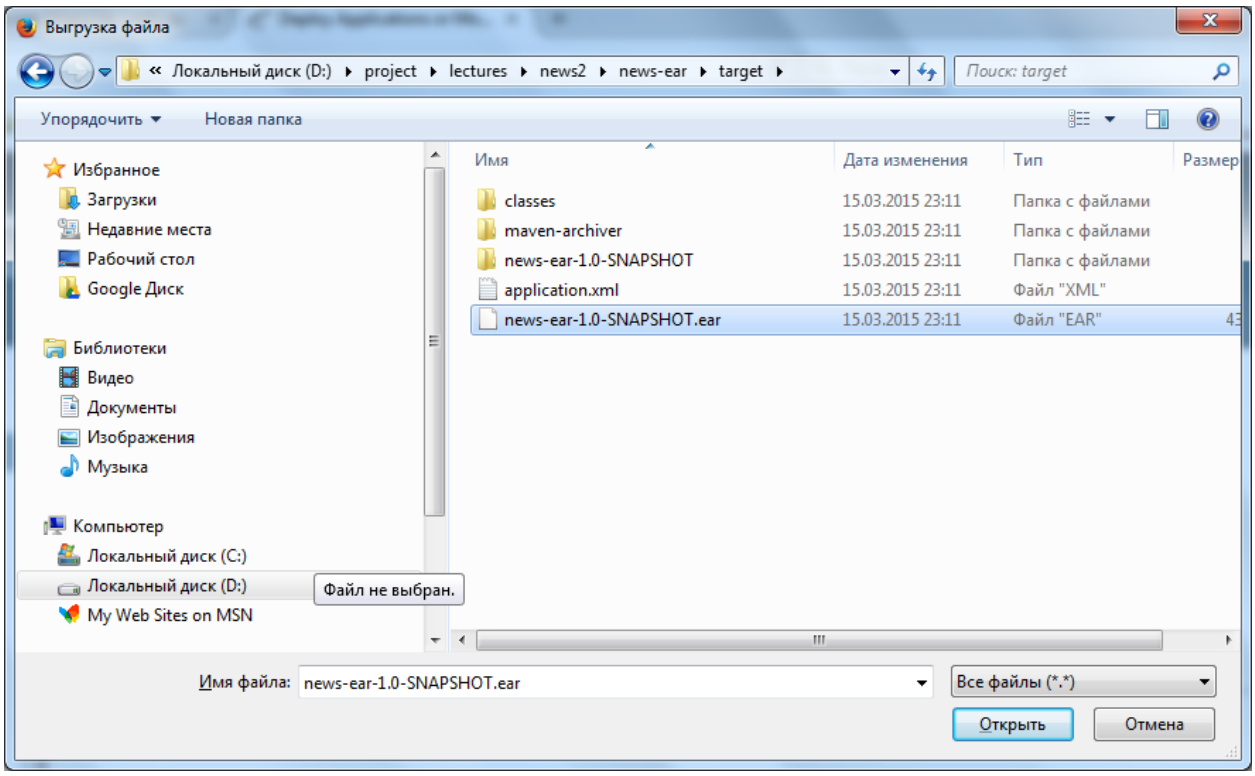
Теперь переходим в JDBC/JDBC Resources и создаем ресурс с именем [jdbc/News](), указывающий на этот пул



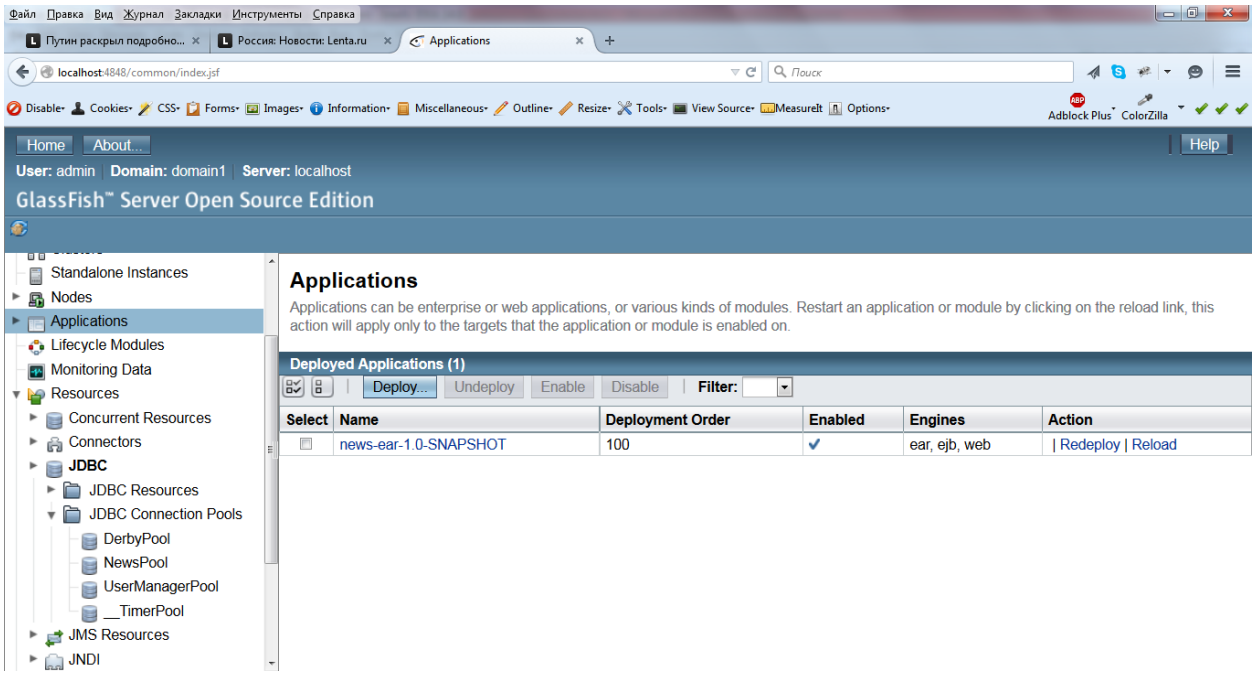
18. Развертывание приложения

В веб-консоли открываем вкладку Applications и нажимаем Deploy

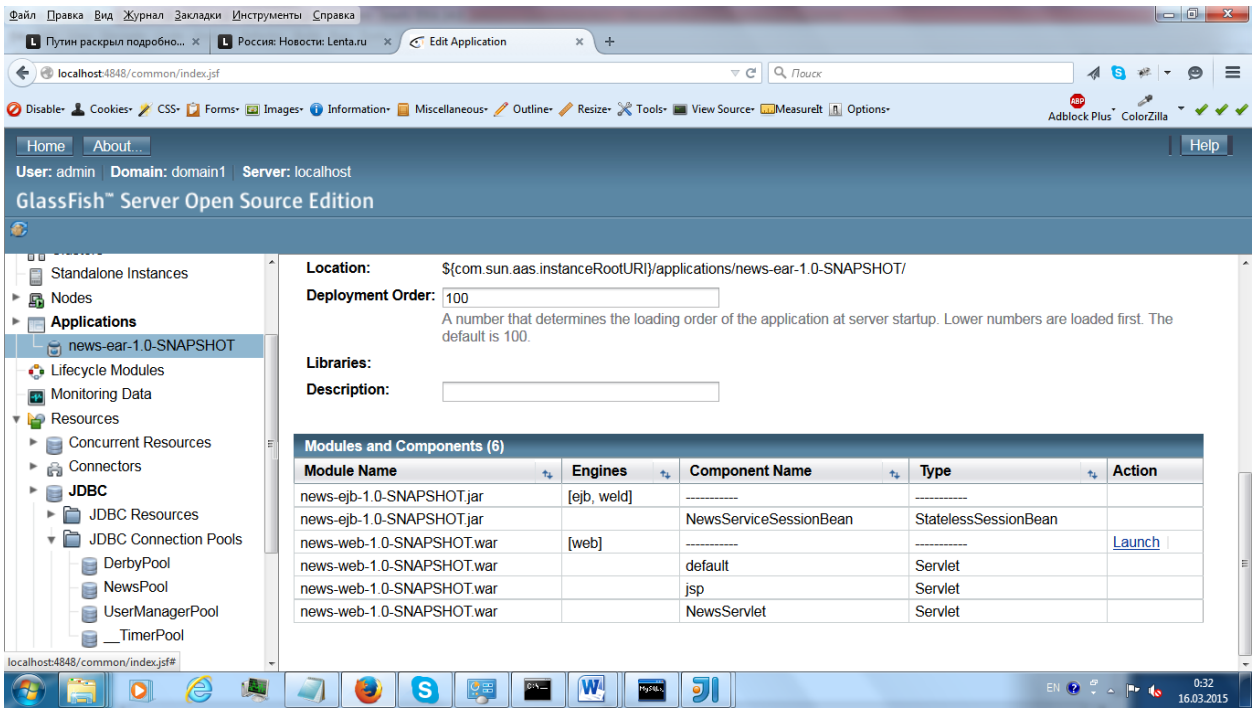
Выбираем ear-файл из папки target проекта news-ear



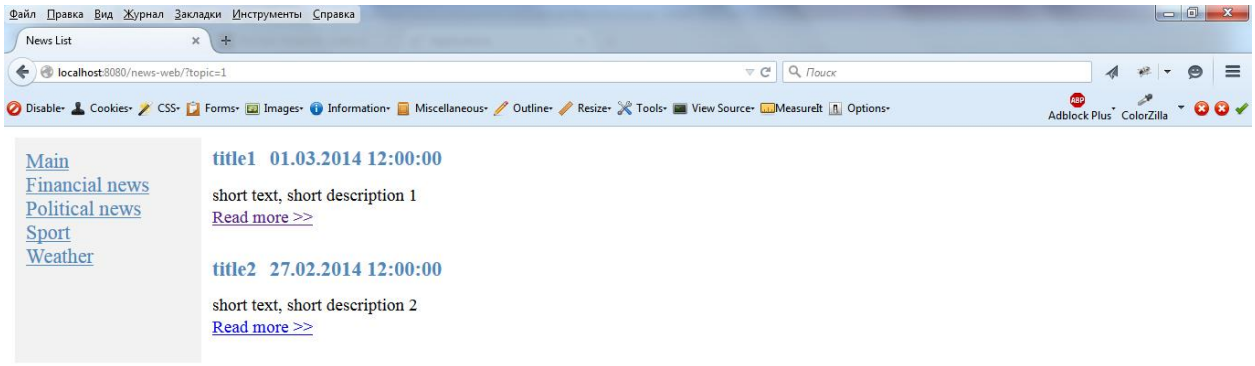
и загружаем его



Можем кликнуть по ссылке, и запустить веб-приложение в браузере



или по ссылке http://localhost:8080/news-web



19. Домашнее задание. Сделать возможность добавления новости через отправку веб-формы.

- создать в папке components файл add_news.jsp, в котором создать html-форму для добавления новости
- в файле news.jsp сделать include нового файла
- сделать сервлет AddNewsServlet, переопределить метод doPost, принять данные. Переданные из формы как параметры запроса, выполнить обращение к ejb методу saveNews и сделать редирект на страницу с лентой новостей.
- зарегистрировать сервлет в web.xml-файле

- развернуть приложение и убедиться, что функциональность работает правильно.

20. Дополнительно. Создание Rest-сервиса

20.1 Подключение зависимости для Jersey. В файле pom.xml проекта news-web добавим зависимость для Jersey

```
<dependency>
  <groupId>org.glassfish.jersey.containers</groupId>
  <artifactId>jersey-container-servlet-core</artifactId>
  <version>2.16</version>
  <scope>provided</scope>
</dependency>
```

В web-xml регистрируем сервлет для jersey

```
<servlet>
  <servlet-name>NewsServiceApplication</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>javax.ws.rs.Application</param-name>
    <param-value>com.practice.news.servlet.service.rest.NewsServiceRestApplication</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>


<servlet-mapping>
  <servlet-name>NewsServiceApplication</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

20.2 Создаем пакет в web-проекте **com.practice.news.servlet.service.rest** и классы в нем:

NewsServiceRest – собственно, сам endpoint сервиса
NewsServiceRestApplication – служебный класс, реализации которого требует jersey.

В сервисе реализуем операции, например, получения новостей, устанавливаем пути и параметры сервиса с помощью аннотаций jax-ws (см. реализацию).

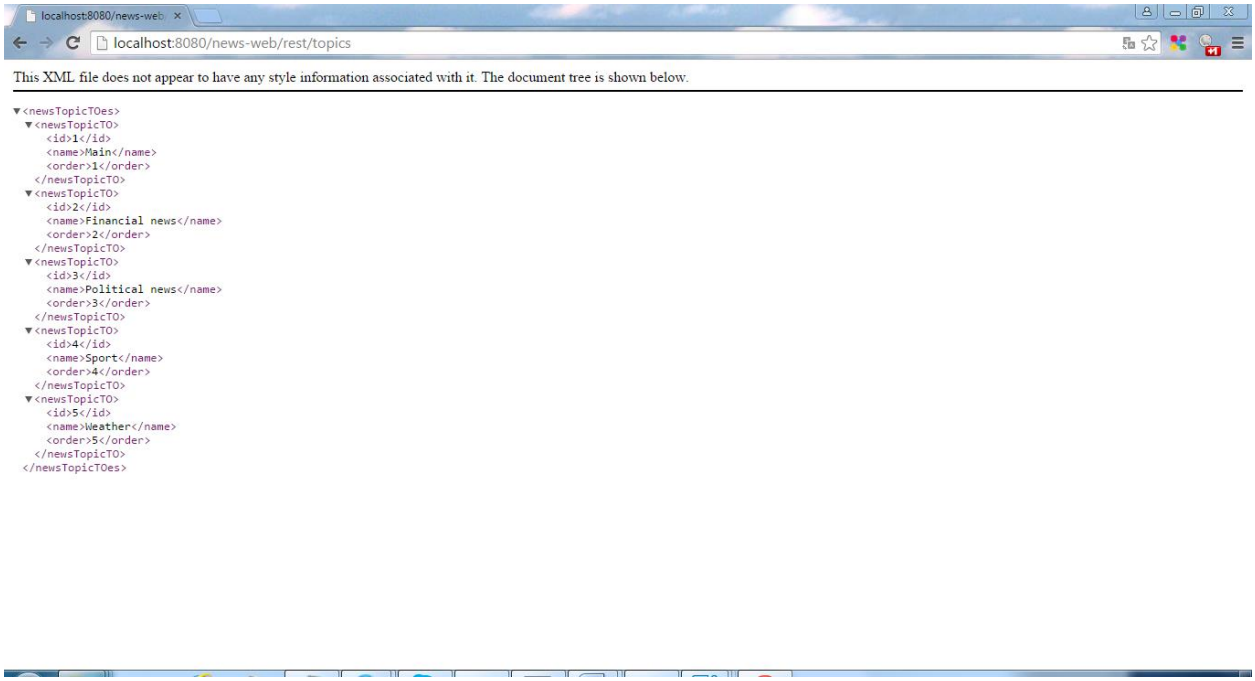
Т.к. реализованный нами сервис не является сервлетом в обычном понимании, приходится выполнять получение экземпляра ejb-компонента через InitialContext.

```
private NewsService getNewsService() {
  try {
    Context ctx = new InitialContext();
    return (NewsService) ctx.lookup(NewsService.BEAN_NAME);
  } catch (NamingException e) {
    throw new WebApplicationException(Response.Status.INTERNAL_SERVER_ERROR);
```

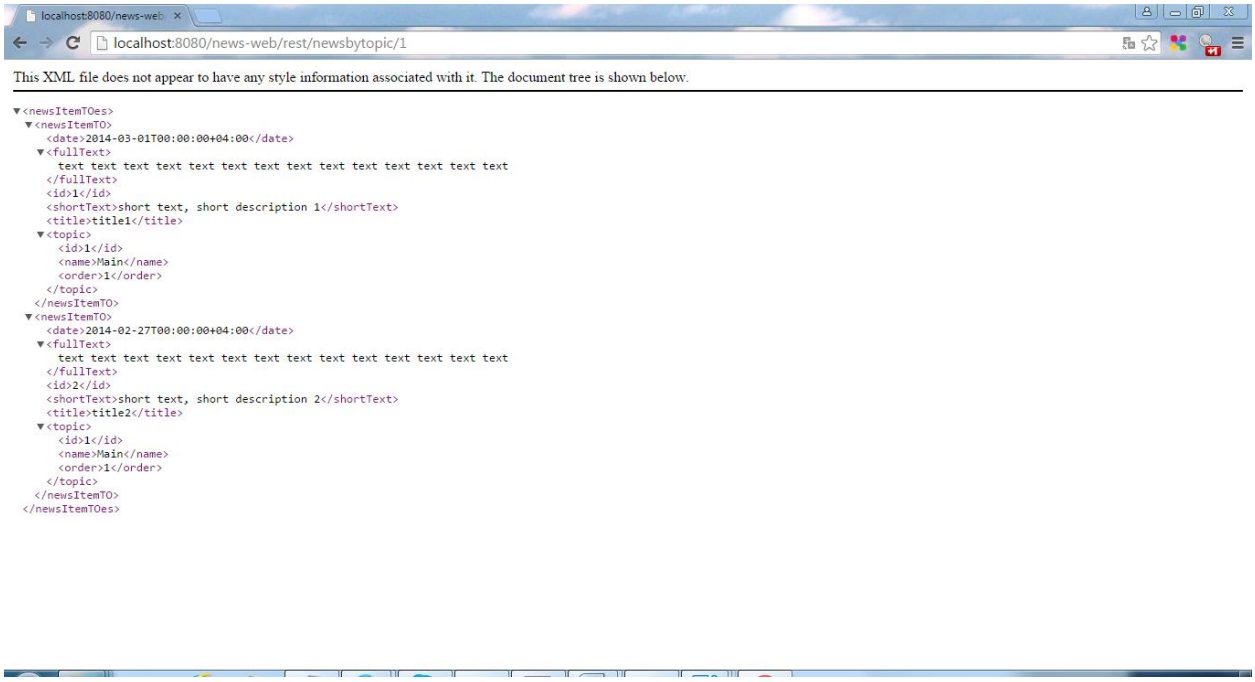
```
        }
    }
```

Для классов доменной модели NewsItemTO, NewsTopicTO добавляем аннотации jaxb @XmlRootElement

Собираем проект, вновь разворачиваем его на сервере
Проверяем работоспособность сервиса. Для REST достаточно просто отправить запрос к нужному URL
например,
http://localhost:8080/news-web/rest/topics

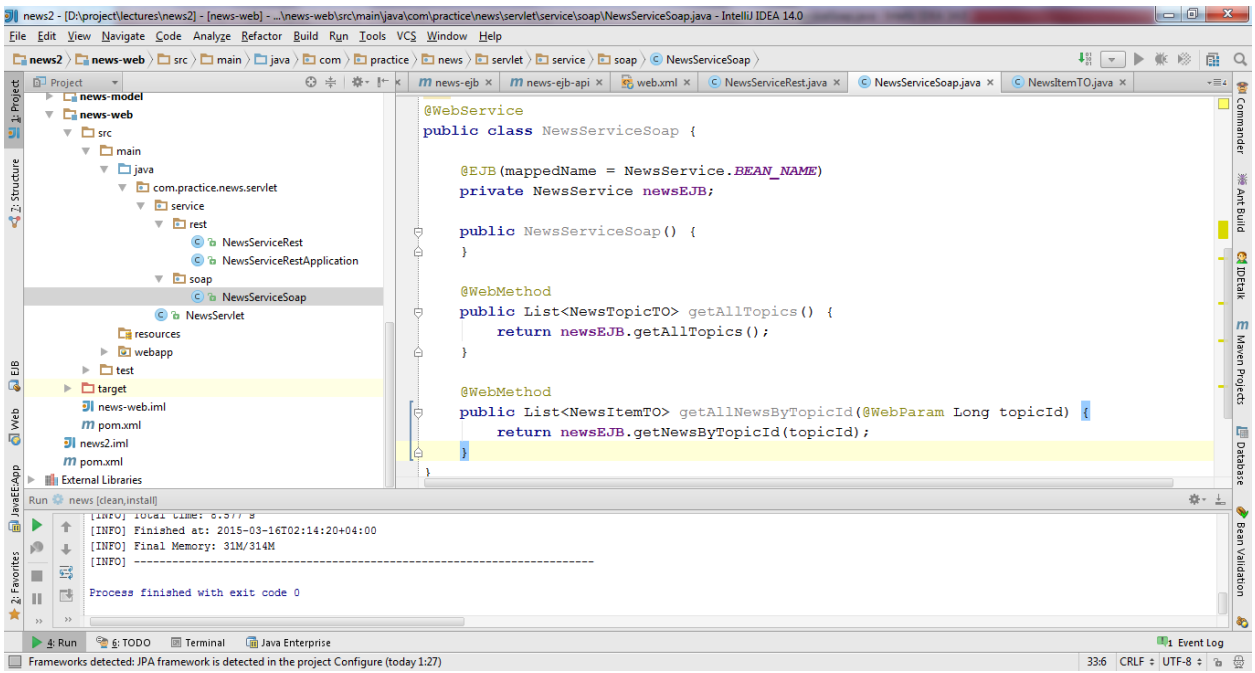


http://localhost:8080/news-web/rest/newsbytopic/2



21. Дополнительно. Создание SOAP-сервиса

21.1 В проекте news-web добавим пакет com.practice.news.servlet.service.soap и создадим в нем класс **NewsServiceSoap.**



Повесим на класс аннотацию @WebService, методы отметим аннотациями @WebMethod,
В рамках использования аннотации @WebService допустимо пользоваться Dependency Injection. Поэтому доступ к EJB можно получить с помощью аннотации.

Добавим в web.xml маппинг для класса сервиса как для сервлета

```
<!--SOAP Service mapping-->
```

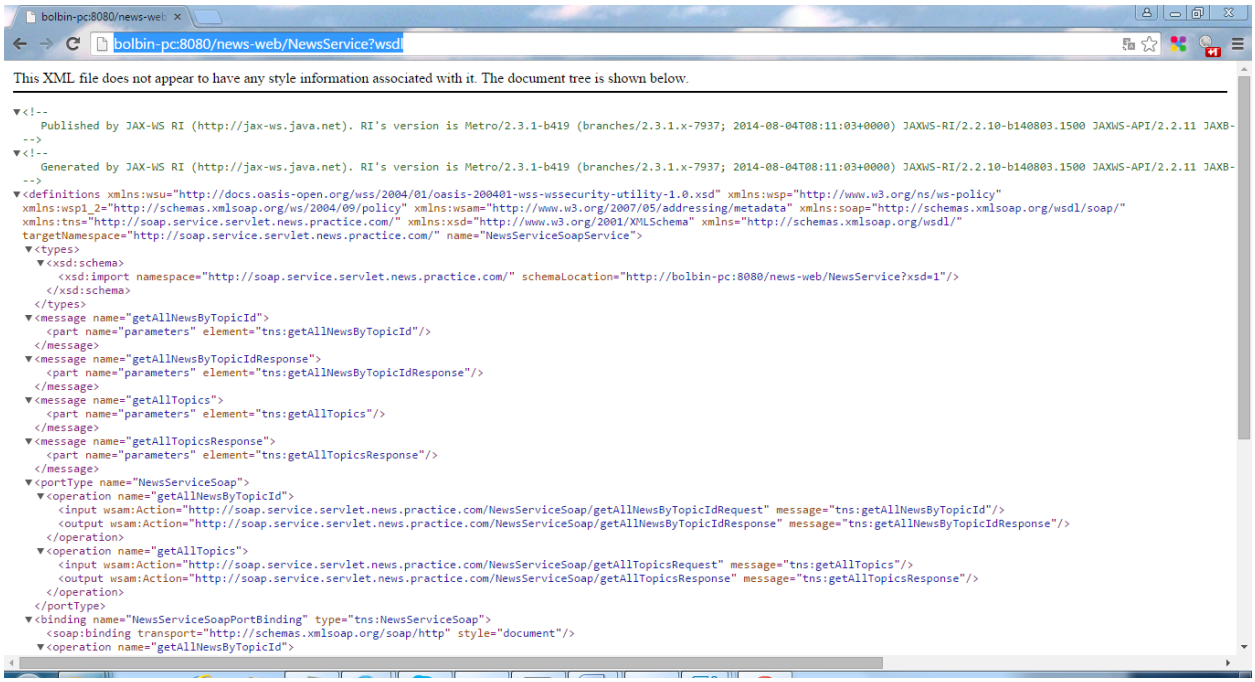
```
<servlet>
   <servlet-name>WebServiceServlet</servlet-name>
   <servlet-class>com.practice.news.servlet.service.soap.NewsServiceSoap</servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
   <servlet-name>WebServiceServlet</servlet-name>
   <url-pattern>/NewsService</url-pattern>
</servlet-mapping>
```

Пересоберем и передеплоим приложение

Откроем ссылку

http://bolbin-pc:8080/news-web/NewsService?wsdl

И проверим содержимое wsdl-документа сервиса



Теперь можем написать клиентское приложение

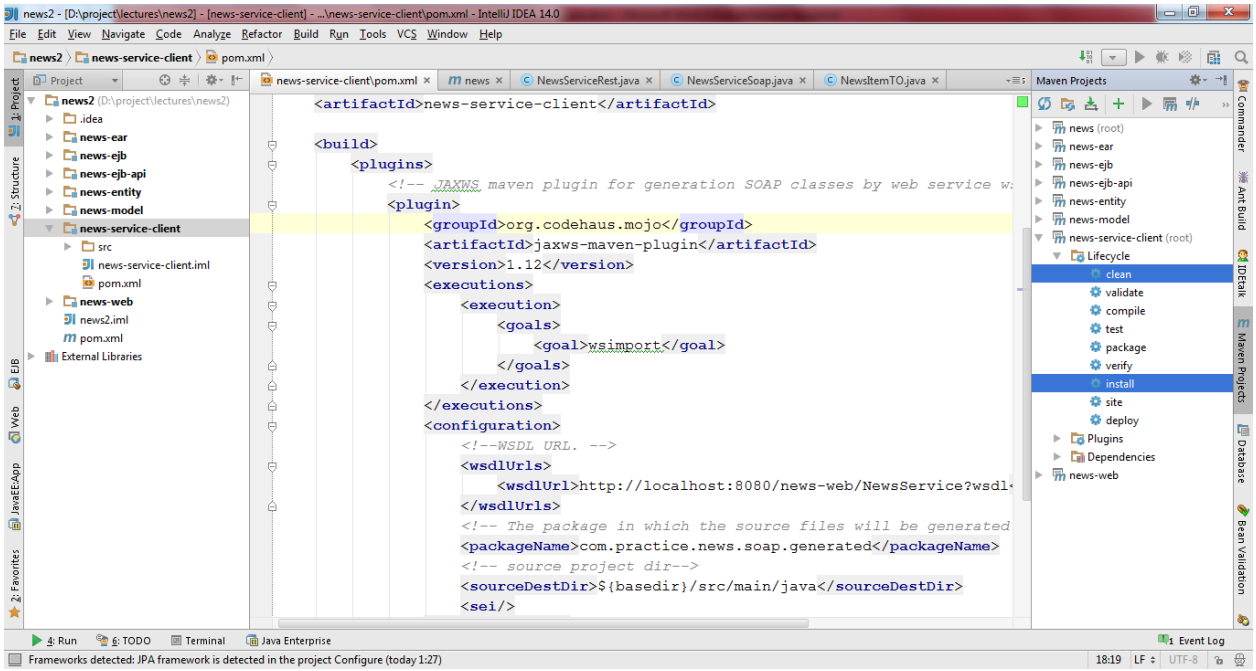
Создаем новый модуль **news-service-client**. В его pom-файле подключаем плагин для генерации клиентского кода web-сервиса по wsdl

```
<build>
   <plugins>
      <!-- JAXWS maven plugin for generation SOAP classes by web service wsdl.-->
      <plugin>
         <groupId>org.codehaus.mojo</groupId>
         <artifactId>jaxws-maven-plugin</artifactId>
         <version>1.12</version>
         <executions>
            <execution>
               <goals>
                  <goal>wsimport</goal>
               </goals>
            </execution>
```

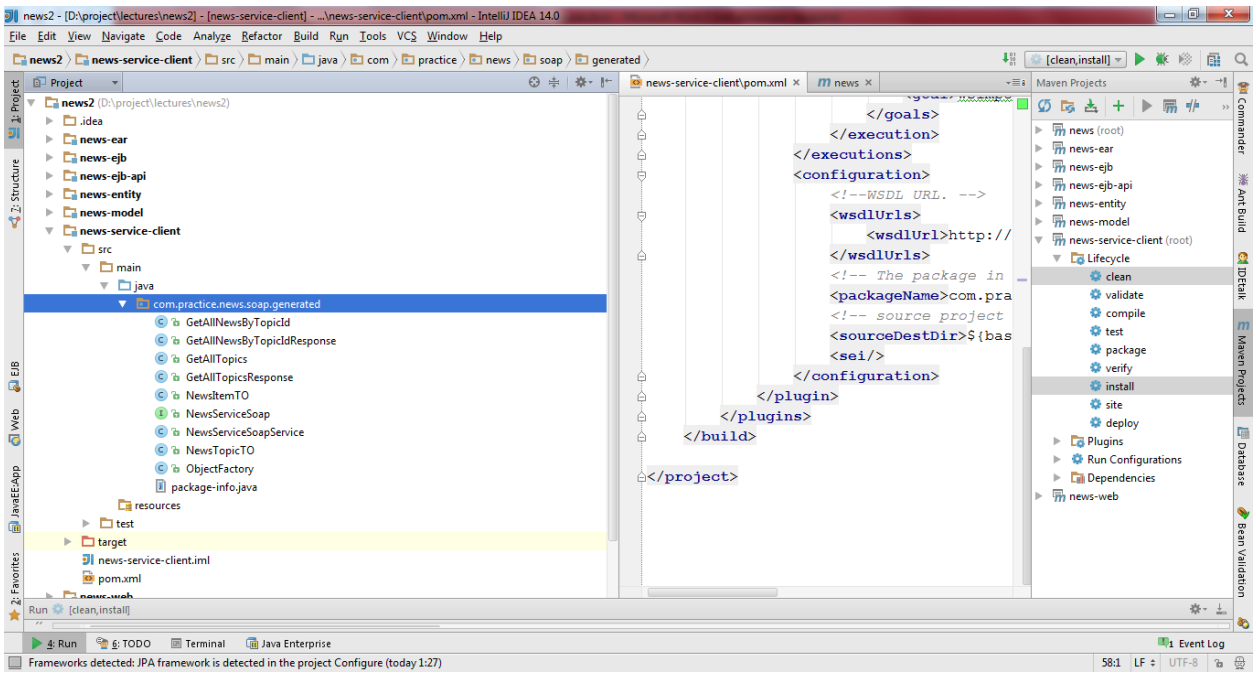
```
        </executions>
        <configuration>
          <!--WSDL URL. -->
          <wsdlUrls>
            <wsdlUrl>http://localhost:8080/news-web/NewsService?wsdl</wsdlUrl>
          </wsdlUrls>
          <!-- The package in which the source files will be generated. -->
          <packageName>com.practice.news.soap.generated</packageName>
          <!-- source project dir-->
          <sourceDestDir>${basedir}/src/main/java</sourceDestDir>
          <sei/>
        </configuration>
      </plugin>
    </plugins>
</build>
```

В главном pom-файле корневого проекта я не стал добавлять клиент как модуль. Обратите внимание, т.к. клиентский код будет сгенерирован по wsdl сервиса, важно, чтобы в момент сборки проекта **news-service-client** глассфиш был запущен и приложение было развернуто на нем.

Собираем клиентское приложение news-service-client



В коде этого проекта появляются сгенерированные классы

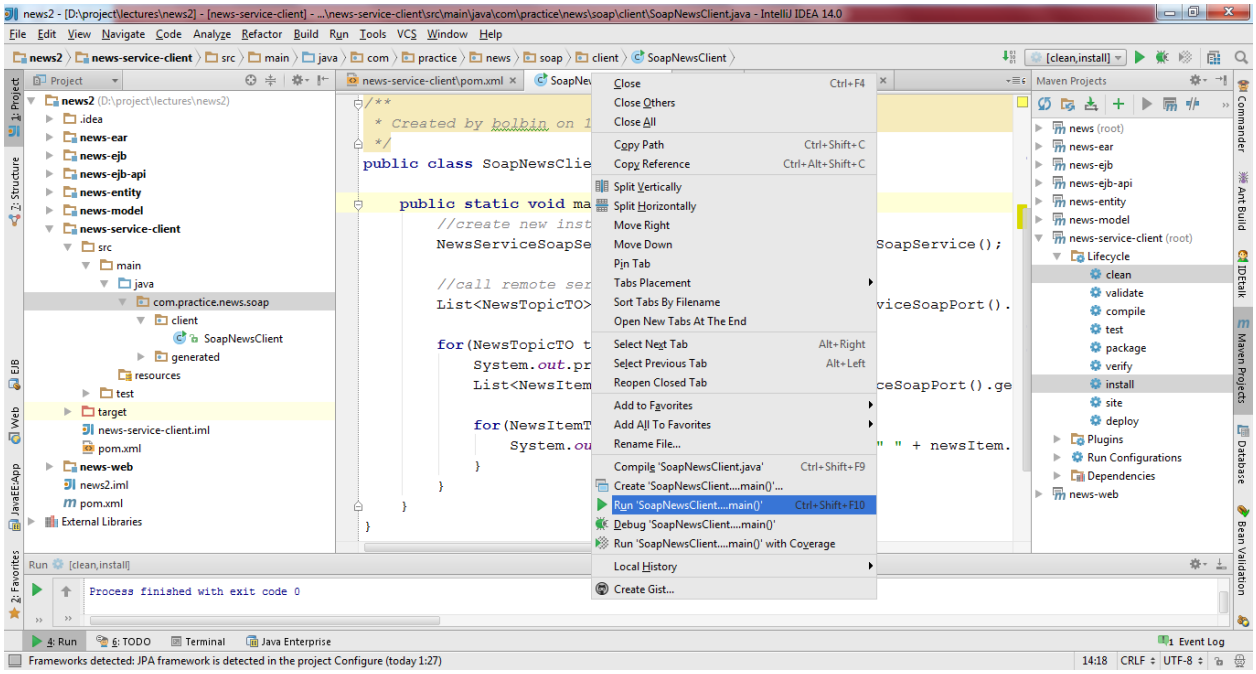


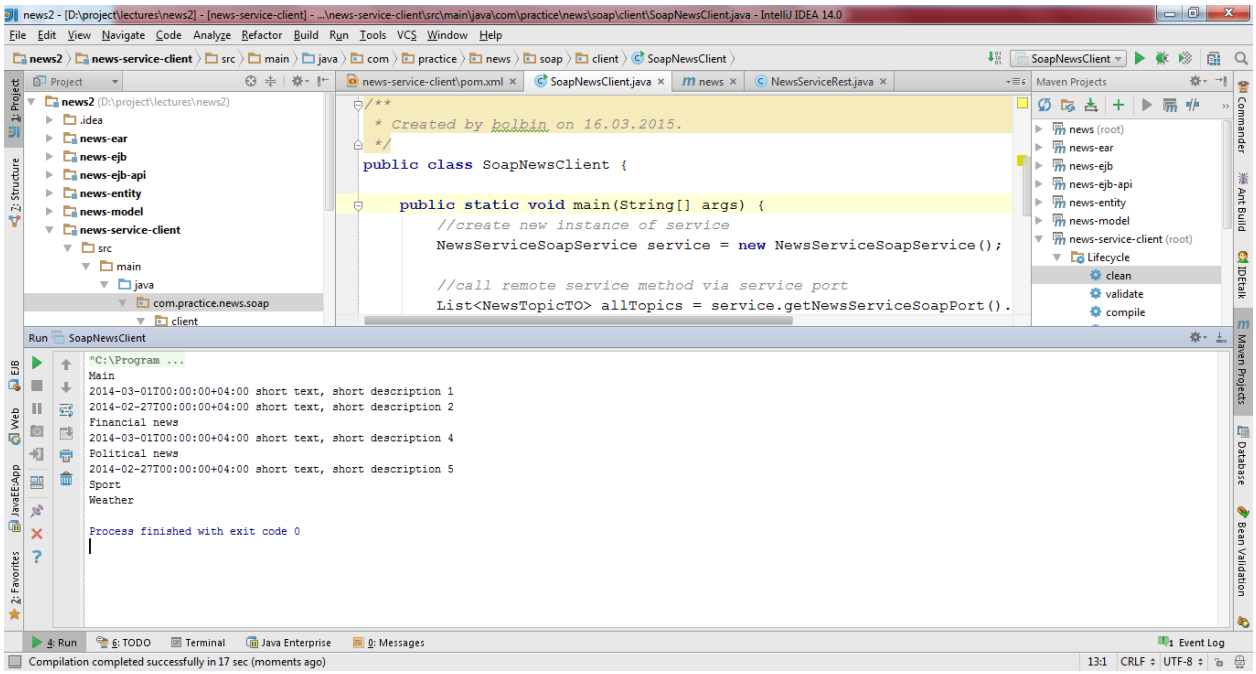
теперь напишем свой java-класс для тестирования обращения к веб-сервису

(**SoapNewsClient.java**)

```
public class SoapNewsClient {

   public static void main(String[] args) {
      //create new instance of service
      NewsServiceSoapService service = new NewsServiceSoapService();

      //call remote service method via service port
      List<NewsTopicTO> allTopics = service.getNewsServiceSoapPort().getAllTopics();

      for(NewsTopicTO topic : allTopics) {
         System.out.println(topic.getName());
         List<NewsItemTO> news = service.getNewsServiceSoapPort().getAllNewsByTopicId(topic.getId());

         for(NewsItemTO newsItem : news) {
            System.out.println(newsItem.getDate() + " " + newsItem.getShortText());
         }
      }
   }
}
```

Который вызывает методы сервиса через клиентский «порт», и выводит все новости по всем темам. Запустим приложение и проверим работоспособность клиента





Обратите внимание, что сгенерированные на клиентской стороне классы NewsItemTO и NewsTopicTO отличаются от оригинальных.