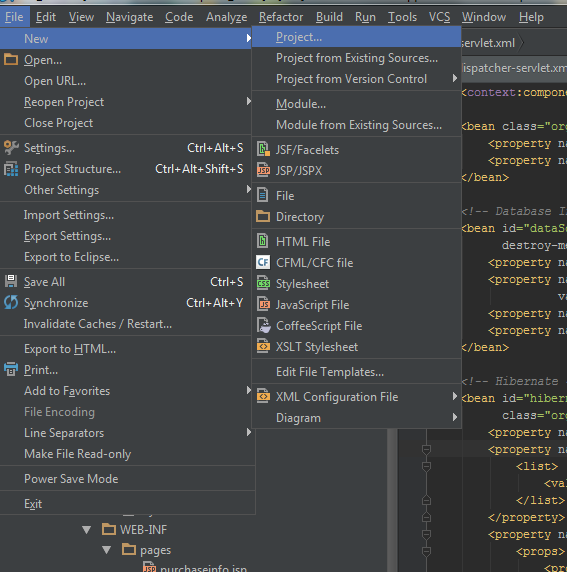
Лабораторная №6

Spring Framework.

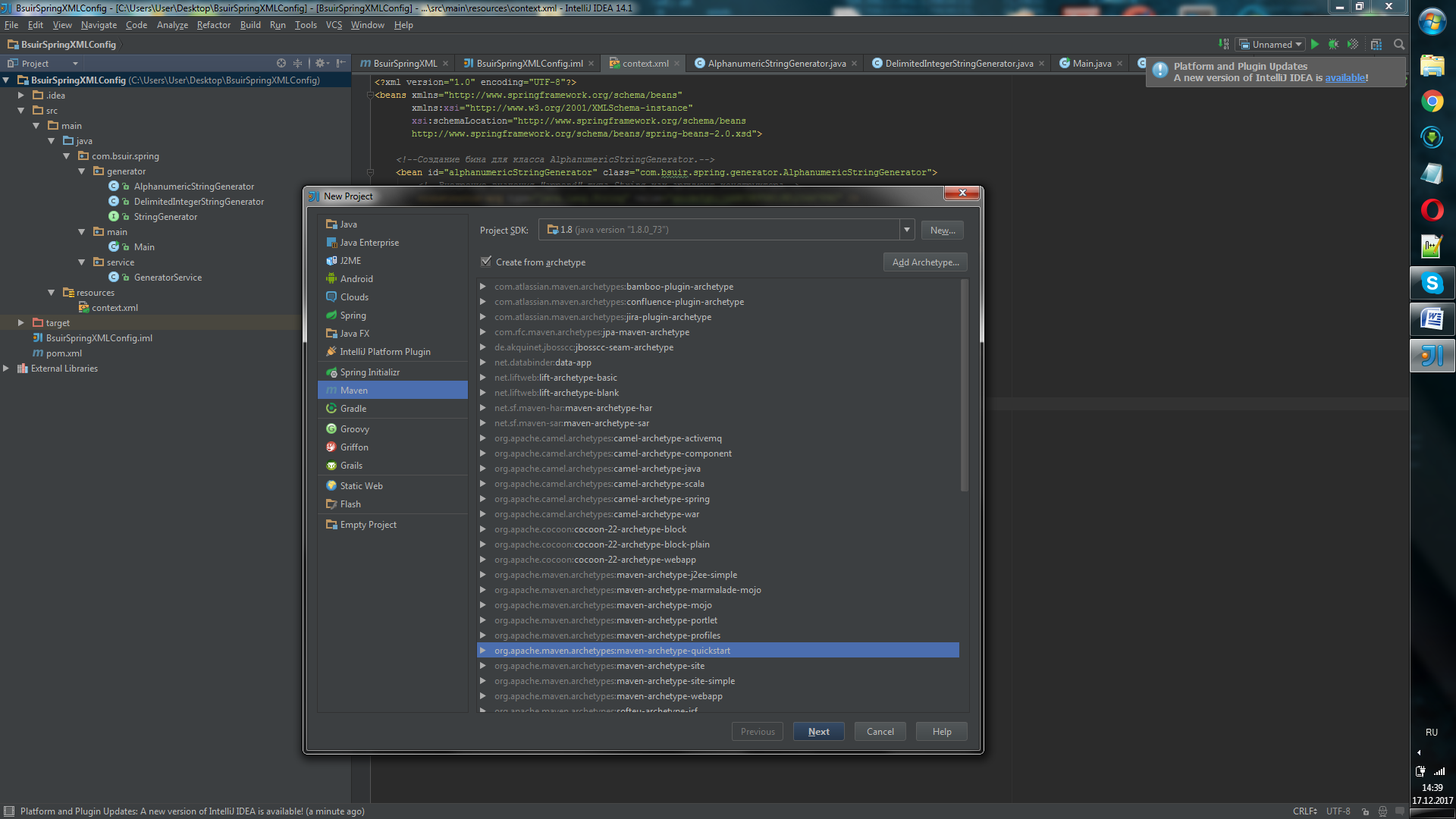
**SpringXML, Spring Annotation**.

Для начала необходимо создать проект в среде разработки (IntelliJ (желательно), NetBeans, Eclipse).



Для этого выбираем File->New->Project

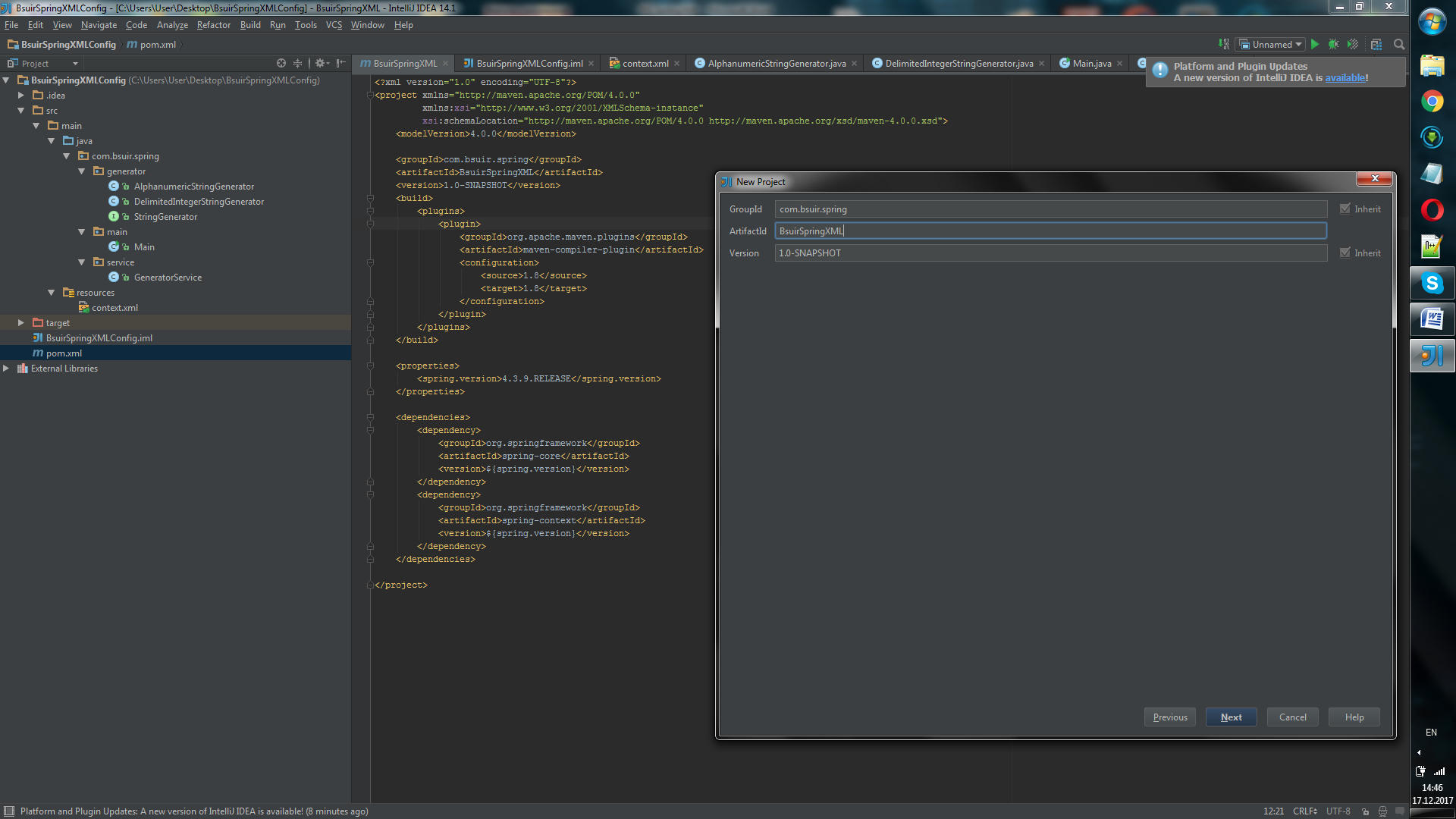
Переходим в Maven и выбираем архетип:



Далее задаем GroupId с указанием доменной зоны и фамилии через точку.

В поле ArtifactId задаем имя проекта.

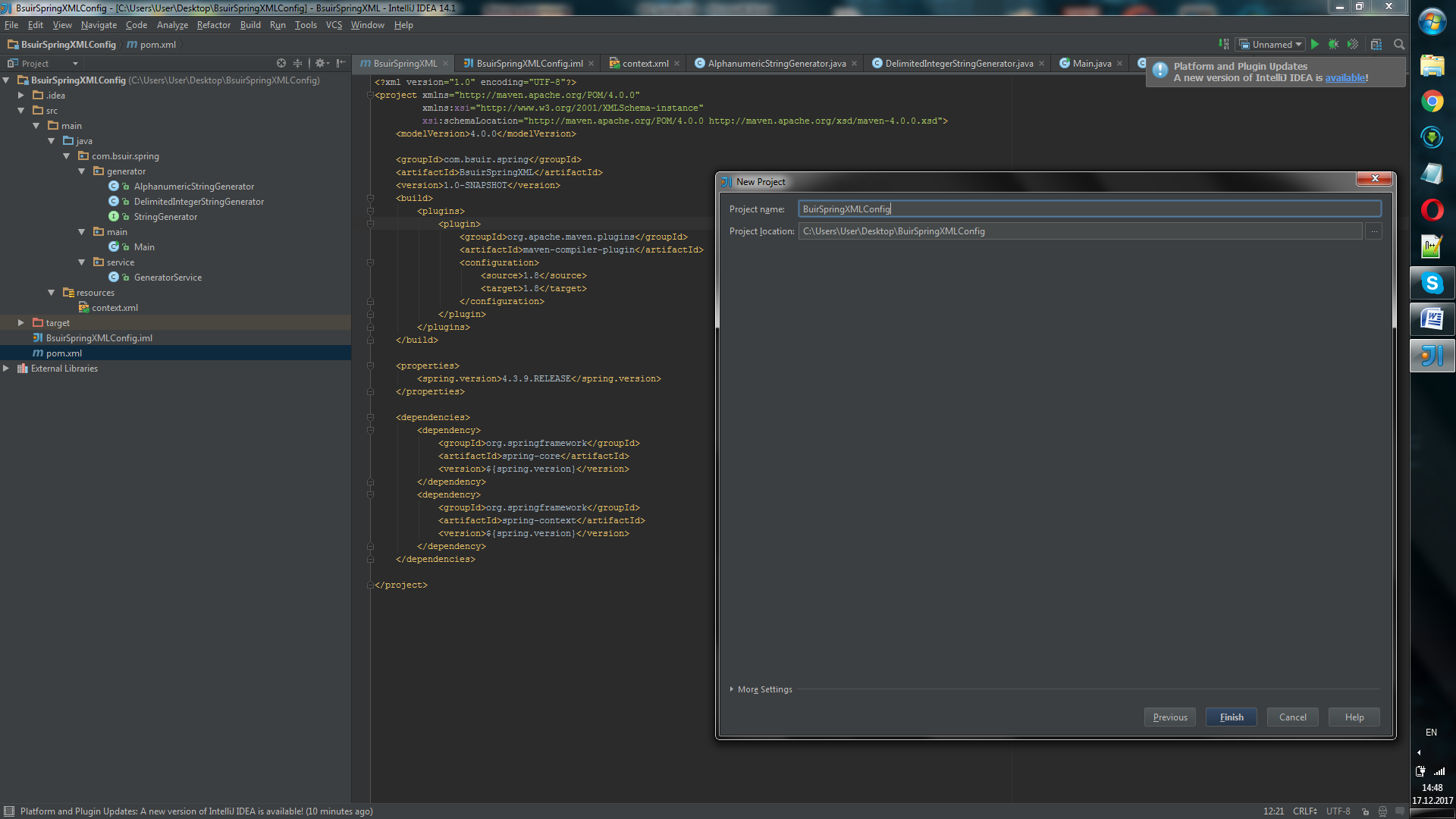
Строку Version оставляем как есть.



Проверяем правильность внесенных данный и подтверждаем.

# 

Задаем имя проекту



Теоритическая часть

# Что такое Spring?

Для начала пара слов, что же такое Spring. В настоящее время, под термином "Spring" часто подразумевают целое семейство проектов. В большинстве своем, они развиваются и курируются компанией Pivotal и силами сообщества. Ключевые (но не все) проекты семейства Spring это:

• Spring Framework (или Spring Core)

Ядро платформы, предоставляет базовые средства для создания приложений — управление компонентами (бинами, beans), внедрение зависимостей, MVC фреймворк, транзакции, базовый доступ к БД. В основном это низкоуровневые компоненты и абстракции. По сути, неявно используется всеми другими компонентами.

• Spring MVC (часть Spring Framework)

Стоит упомянуть отдельно, т.к. мы будем вести речь в основном о веб-приложениях. Оперирует понятиями контроллеров, маппингов запросов, различными HTTP абстракциями и т.п. Со Spring MVC интегрированы нормальные шаблонные движки, типа Thymeleaf, Freemaker, Mustache, плюс есть сторонние интеграции с кучей других. Так что никакого ужаса типа JSP или JSF писать не нужно.

• Spring Data

Доступ к данным: реляционные и нереляционные БД, KV хранилища и т.п.

• Spring Cloud

Много полезного для микросервисной архитектуры — service discovery, трасировка и диагностика, балансировщики запросов, circuit breaker-ы, роутеры и т.п.

• Spring Security

Авторизация и аутентификация, доступ к данным, методам и т.п. OAuth, LDAP, и куча разных провайдеров.

• Spring Integration

Обработка данных из разных источников. Если надо раз в час брать файл с ФТП, разбивать его на строки, которые потом фильтровать, а дальше отправлять в какую-то очередь — это к Spring Integration.

В данной лабораторной работе мы познакомимся ближе со Spring MVC и Spring Security.

**Плюсы Spring**:

• Spring можно использовать для построение любого приложения на языке Java, что выгодно отличает его от многих других платформ (таких как Apache Struts)

• для использования ядра Spring нужно внести минимальные изменения в код приложения (принцип философии Spring – минимальное воздействие)

• Spring является модульной средой и позволяет использовать отдельные свои части без необходимости вводить остальные

• Возможность работы с POJO (без контейнеров EJB)

• Существует большое количество расширений Spring для построения приложений на Java Enterprise платформе

• Сообщество Spring – одно из лучших сообществ из всех проектов с открытым исходным кодом, списки рассылки и форумы всегда активны.

• У Spring отличная подробная документация • Spring активно развивается

**Spring** обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе [Java](https://ru.wikipedia.org/wiki/Java). Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. **Spring** не всецело связан с платформой [Java Enterprise](https://ru.wikipedia.org/wiki/J2EE), несмотря на его масштабную интеграцию с ней, что является важной причиной его популярности.

**Spring**, вероятно, наиболее известен как источник расширений (features), нужных для эффективной разработки сложных бизнес-приложений вне тяжеловесных программных моделей, которые исторически были доминирующими в промышленности. Ещё одно его достоинство в том, что он ввел ранее неиспользуемые функциональные возможности в сегодняшние господствующие методы разработки, даже вне платформы Java.

Этот фреймворк предлагает последовательную модель и делает её применимой к большинству типов приложений, которые уже созданы на основе платформы Java. Считается, что **Spring** реализует модель разработки, основанную на лучших стандартах индустрии, и делает её доступной во многих областях Java.

**Возможности Spring:**

• Использование внедрения зависимостей (DI)

• Поддержка аспектно-ориентированного программирования (в том числе интеграция с AspectJ)

• Язык выражений Spring (SpEL) – позволяет приложению манипулировать объектами Java во время выполнения.

• Встроенная поддержка Bean Validation API – позволяет один раз описать логику проверки достоверности данных и использовать ее как в пользовательском интерфейсе, так и на уровне работы с БД.

• Spring обеспечивает великолепную интеграцию с большинством инструментов доступа к данным (JDBC, Hibernate, MyBatis, JDO, JPA и т.п.)

• Поддержка Object to XML Mapping – преобразование компонентов JavaBean в XML и наоборот (как правило, используется для обмена данными с другими системами)

• Интеграция с JEE – внедрение бинов Spring в компоненты EJB.

• MVC на веб-уровне

• Поддержка электронной почты

• Поддержка планирования заданий

• Поддержка динамических сценариев (Groovy, JRuby, BeanShell)

# Spring Annotation

В начале был EJB 2.1, с его огромным количеством XML-файлов везде где только можно. Не будет особым преувеличением, если сказать, что на одну строку кода для бизнес-логики нужно было написать по крайней мере 10 строк кода от фреймворка и две страницы XML. Локальные и удалённые интерфейсы, ручной JNDI-lookup, многоуровневые try-catch, проверки на RemoteException. Потом создали [Spring framework](http://springframework.org/). Прошло время и Sun осознали урок. EJB 3.0 был даже проще Spring, XML-free, с аннотациями, dependency injection. 3.1 стал ещё одним огромным шагом в сторону упрощения.

По логике, EJB сейчас можно рассматривать как часть того, что предлагает Spring, почему нет реализации EJB в plain Spring, учитывая его поддержку из-коробки JPA 1.0/2.0, JSR-250, JSR-330, JAX-WS/RS и прочего. Spring framework сегодня воспринимается как медленный, тяжёлый и сложный для поддержки фреймворк, в-основном из-за XML-дескрипторов.

Уход от XML рассмотрим во втором примере Теоретической части.

Аннотации:

**@Component** - Аннотация для любого компонента фреймворка.

**@Service** - (Сервис-слой приложения) Аннотация, объявляющая, что этот класс представляет собой сервис – компонент сервис-слоя.

Сервис является подтипом класса @Component. Использование данной аннотации позволит искать бины-сервисы автоматически.

**@Repository** - (Доменный слой) Аннотация показывает, что класс функционирует как репозиторий и требует наличия прозрачной трансляции исключений. Преимуществом трансляции исключений является то, что слой сервиса будет иметь дело с общей иерархией исключений от Spring (DataAccessException) вне зависимости от используемых технологий доступа к данным в слое данных.

**@Controller** - (Слой представления) Аннотация для маркировки java класса, как класса контроллера. Данный класс представляет собой компонент, похожий на обычный сервлет (HttpServlet) (работающий с объектами HttpServletRequest и HttpServletResponse), но с расширенными возможностями от Spring Framework.

**@ResponseBody** - Аннотация показывает что данный метод может возвращать кастомный объект в виде xml, json...

**@RestController** - Аннотация аккумулирует поведение двух аннотаций @Controller и @ResponseBody

**@Transactional** - Перед исполнением метода помеченного данной аннотацией начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании RuntimeException откатывается.

**@Autowired** - Аннотация позволяет автоматически установить значение поля.

**@RequestMapping** - Аннотация используется для маппинга урл-адреса запроса на указанный метод или класс. Можно указывать конкретный HTTP-метод, который будет обрабатываться (GET/POST), передавать параметры запроса.

**@ModelAttribute** - Аннотация, связывающая параметр метода или возвращаемое значение метода с атрибутом модели, которая будет использоваться при выводе jsp-страницы.

**@PathVariable** - Аннотация, которая показывает, что параметр метода должен быть связан с переменной из урл-адреса.

**@Scope** - Аннотация для установки области жизни бина: *singleton* (только один экземпляр бина создается для IoC контейнера; значение по умолчанию ), *prototype* (создается новый экземпляр бина когда приходит запрос на его создание),*request* (один экземпляр бина для каждого HTTP запроса ), *session* (один экземпляр бина для каждой сессии), *globalSession* (один экземпляр бина для каждой глобальной сессии)

**@PostConstruct** - Аннотация для метода, который будет вызван после вызова конструктора бина.

**@PreDestroy** - Аннотация для метода, который будет вызван перед уничтожением бина.

**@Profile** - Аннотация для создания профилей конфигурации проекта. Может применяться как к бинам так и к конфигурационным классам.

# **Области видимости бинов**

**singleton** - Возвращает один и тот же экземпляр бина на каждый запрос контейнера Spring IoC (по умолчанию).

**prototype** - Создает и возвращает новый экземпляр бина на каждый запрос.

**request** - Создает и возвращает экземпляр бина на каждый HTTP запрос\*.

**session** - Создает и возвращает экземпляр бина для каждой HTTP сессии\*.

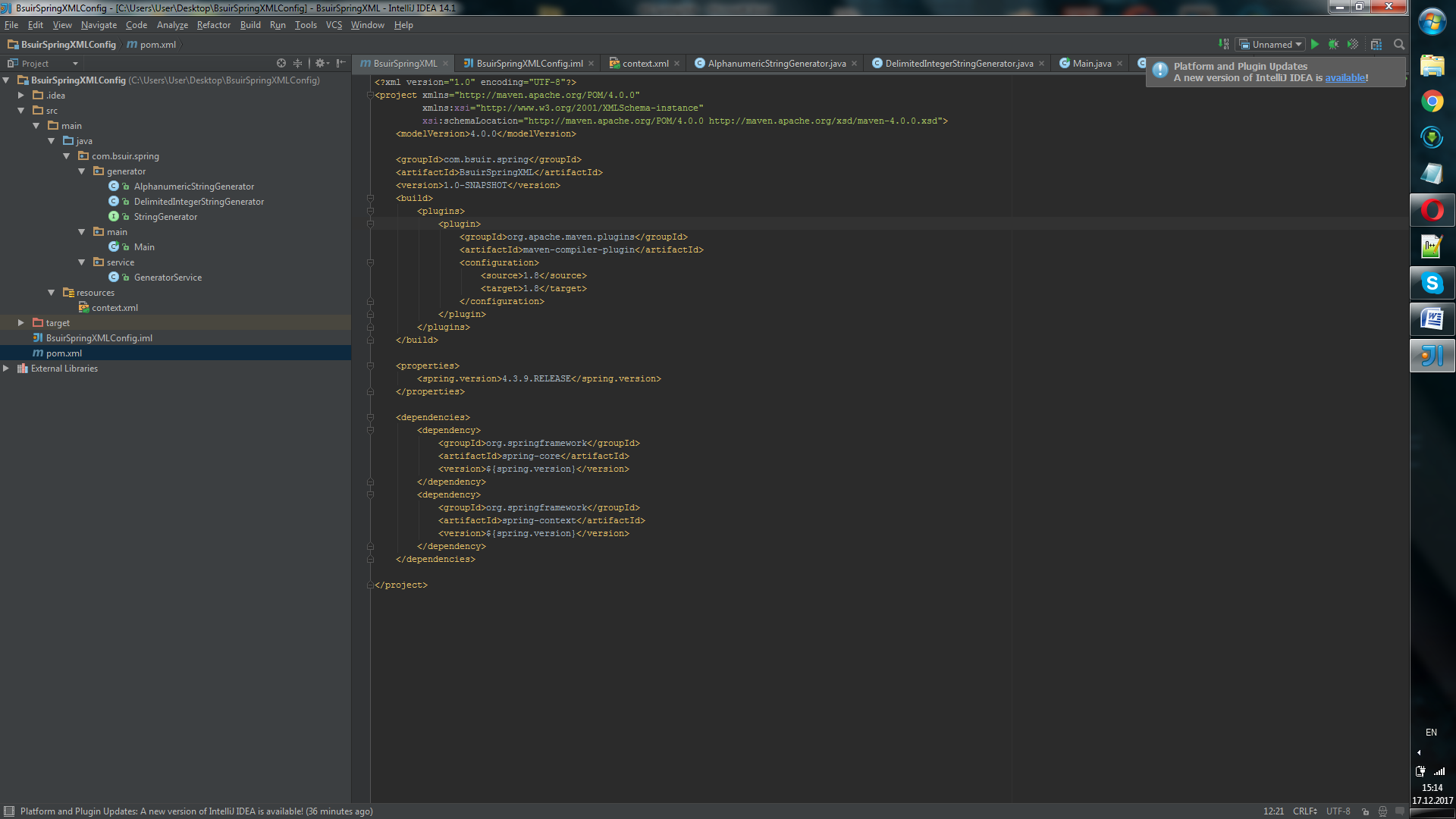
**global-session** - Создает и возвращает экземпляр бина для глобальной HTTP сессии\*.

**Далее рассмотрим различия в использовании XML и Annotation.**

Пример 1 (XML)

Рассмотрим приложение, которое выводит в консоль строки, состоящие из букв и цифр размещенных в рандомном порядке (количество строк, количество символов в строке и интервалы выбирает пользователь)

Структура проекта представлена ниже:



1. В файле **pom.xml** нужно подключить зависимости, необходимые для работы со Spring (**spring-core** и **spring-context**):

<?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">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.bsuir.spring</groupId>  
 <artifactId>BsuirSpringXML</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
 <properties>  
 <spring.version>4.3.9.RELEASE</spring.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 </dependencies>  
  
</project>

Создадим интерфейс StringGenerator единственным методом, который осуществляет генерацию строки.

package com.bsuir.spring.generator;  
  
*/\*\*  
 \* Provides functionality to produce string  
 \*/*public interface StringGenerator {  
  
 String *EMPTY\_STRING* = "";  
  
 */\*\*  
 \* Generates string with random characters.  
 \*  
 \** ***@param*** *count result string length  
 \** ***@return*** *generated string  
 \*/* String generate(int count);  
}

1. Реализуем данный интерфейс двумя способами.

С генерацией сроки букв:

package com.bsuir.spring.generator;  
  
import java.util.Random;  
  
*/\*\*  
 \* Produces an alphanumeric string.  
 \*/*public class AlphanumericStringGenerator implements StringGenerator {  
  
 private static final Random *RANDOM* = new Random();  
  
 private String source;  
  
 public AlphanumericStringGenerator(String source) {  
 this.source = source;  
 }  
  
 @Override  
 public String generate(int count) {  
 if (source == null || source.isEmpty()) {  
 return *EMPTY\_STRING*;  
 }  
 if (count < 0) {  
 throw new IllegalArgumentException("Length of generated string must be positive.");  
 }  
 StringBuilder result = new StringBuilder();  
 for (int index = 0; index < count; index++) {  
 int sourceIndex = *RANDOM*.nextInt(source.length());  
 result.append(source.charAt(sourceIndex));  
 }  
 return result.toString();  
 }  
}

С генерацией сроки цифр:

package com.bsuir.spring.generator;  
  
import java.util.Random;  
  
*/\*\*  
 \* Produces a string full of positive integers.  
 \*/*public class DelimitedIntegerStringGenerator implements StringGenerator {  
  
 private static final Random *RANDOM* = new Random();  
  
 private String delimiter;  
 private int endInclusive;  
  
 public DelimitedIntegerStringGenerator(String delimiter, int endInclusive) {  
 this.delimiter = delimiter;  
 this.endInclusive = endInclusive;  
 }  
  
 @Override  
 public String generate(int count) {  
 if (endInclusive <= 0) {  
 throw new IllegalArgumentException("Property endInclusive must be positive integer.");  
 }  
 if (count < 0) {  
 throw new IllegalArgumentException("Length of generated string must be positive.");  
 }  
 StringBuilder result = new StringBuilder();  
 for (int index = 0; index < count; index++) {  
 result.append(*RANDOM*.nextInt(endInclusive)).append(delimiter);  
 }  
 return result.deleteCharAt(count - 1).toString();  
 }  
}

Создадим класс-сервис, осуществляющий генерацию текста и цифр, а так же содержащий гетеры и сеттеры.

package com.bsuir.spring.service;  
  
import com.bsuir.spring.generator.StringGenerator;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class GeneratorService {  
  
 private StringGenerator stringGenerator;  
  
 private int stringsCount;  
 private int length;  
  
 // Default constructor is needed if dependency injection is going to be performed via setters  
 public GeneratorService() {  
 }  
  
 public GeneratorService(StringGenerator stringGenerator, int stringsCount, int length) {  
 this.stringGenerator = stringGenerator;  
 this.stringsCount = stringsCount;  
 this.length = length;  
 }  
  
 public List<String> generateStrings() {  
 List<String> strings = new ArrayList<>();  
 for (int count = 0; count < stringsCount; count++) {  
 strings.add(stringGenerator.generate(length));  
 }  
 return strings;  
 }  
  
 public StringGenerator getStringGenerator() {  
 return stringGenerator;  
 }  
  
 public int getStringsCount() {  
 return stringsCount;  
 }  
  
 public void setStringGenerator(StringGenerator stringGenerator) {  
 this.stringGenerator = stringGenerator;  
 }  
  
 public void setStringsCount(int stringsCount) {  
 this.stringsCount = stringsCount;  
 }  
  
 public int getLength() {  
 return length;  
 }  
  
 public void setLength(int length) {  
 this.length = length;  
 }  
}

Создадим бины, соответствующие данным классам и сконфигурируем их. Опишем конфигурацию в файле string.xml.

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">  
  
 *<!--Создание бина для класса AlphanumericStringGenerator.-->* <bean id="alphanumericStringGenerator" class="com.bsuir.spring.generator.AlphanumericStringGenerator">  
 *<!--Внедрение значения "append" типа String как аргумент конструктора-->* <constructor-arg type="java.lang.String" value="abcdefghijkABCDEFGHIJK123467890" />  
 </bean>  
  
 *<!--Создание бина для класса DelimitedIntegerStringGenerator.-->* <bean id="delimitedIntegerStringGenerator" class="com.bsuir.spring.generator.DelimitedIntegerStringGenerator">  
 *<!--Внедрение полей класса при помощи сеттеров-->* <constructor-arg type="java.lang.String" value="," />  
 <constructor-arg type="int" value="10" />  
 </bean>  
  
 *<!--Создание бина для класса GeneratorService.-->* <bean id="generatorService" class="com.bsuir.spring.service.GeneratorService">  
 *<!--Внедрение бина с id "alphanumericStringGenerator" как поле stringGenerator с использованием сеттера-->* <property name="stringGenerator" ref="delimitedIntegerStringGenerator"/>  
 *<!--Внедрение значения 20 в поле "text" с использованием сеттера-->* <property name="stringsCount" value="1" />  
 <property name="length" value="5" />  
 </bean>  
  
</beans>

Для внедрения свойств через конструктор используется тег **<constructor-arg />**. При использовании этого метода внедрения в классе, для которого создается бин, происходит вызов конструктора с параметрами тех типов, которые заданы в атрибуте **type**. Так, например, при генерировании первого бина будет вызван конструктор AlphanumericStringGenerator(), куда будет передано значение «abcdefghijkABCDEFGHIJK123467890». Отметим, что тегов **<constructor-arg />** для одного бина может быть объявлено несколько: произойдет вызов конструктора с несколькими параметрами. В то же время, если ни одного тега **<constructor-arg />** не задано, будет вызван конструктор без параметров (как, например, в третьем бине).

При внедрении значений через свойства используется тег **<property />**, где определяется название поля для внедрения и значение, которое необходимо передать в метод **set**, который должен иметь атрибут доступа public. Так в бине с id «generatorService» вначале с использованием конструктора без параметров будет создан экземпляр класса GeneratorService, а затем будут вызваны методы setStringGenerator, куда будет передана ссылка на бин с id «delimitedIntegerStringGenerator», и методы setStringsCount со значением параметра «1» и setLength со значением 5.

Также заметим, что для внедрения стандартных типов можно использовать атрибут **value**, куда передается строковое значение, в дальнейшем преобразуемое в объект необходимого типа. Если же необходимо внедрить ссылку на бин контекста необходимо использовать атрибут **ref**, куда передать идентификатор бина.

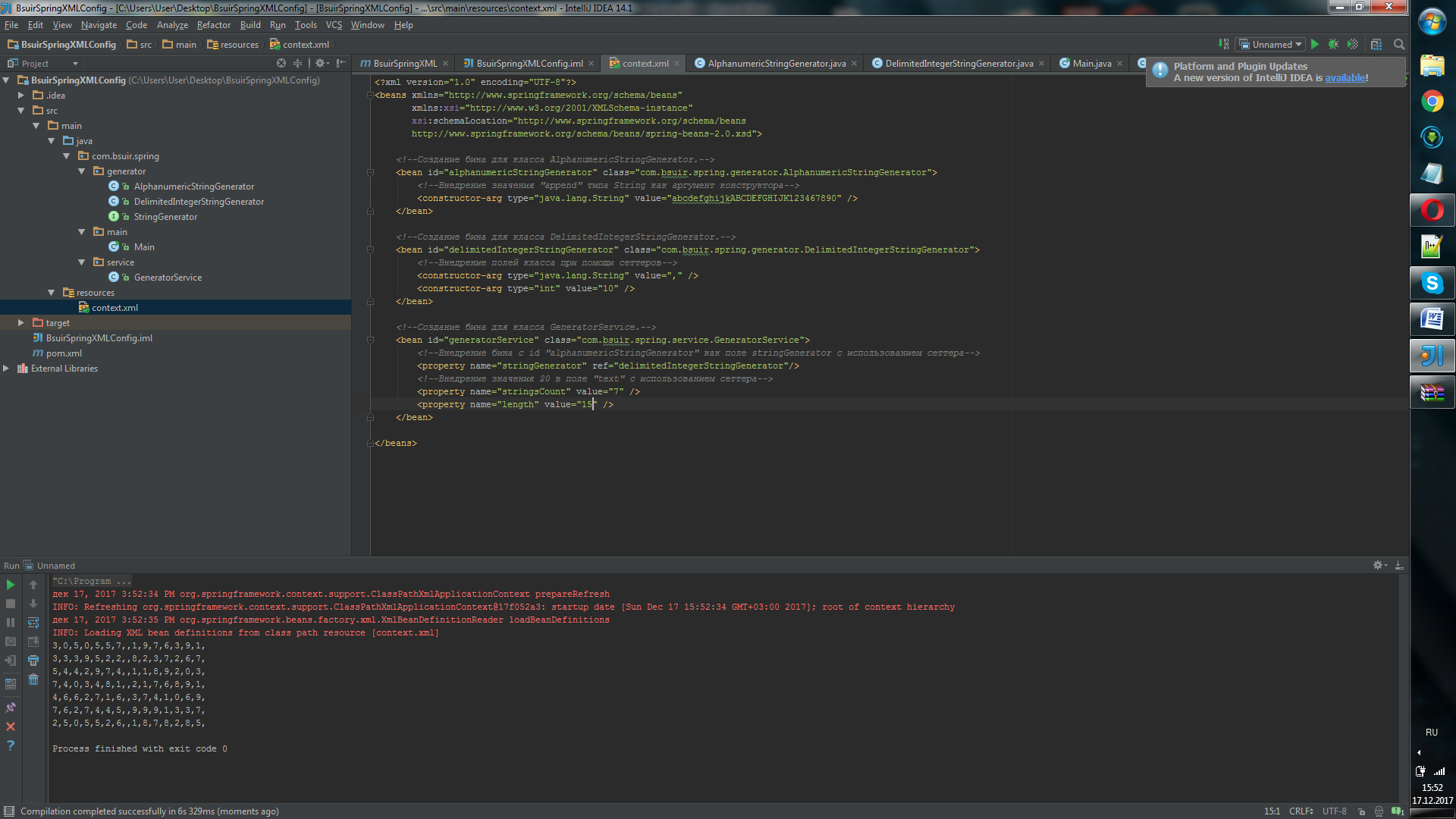
Ещё одним способом внедрения зависимостей является использование фабричного метода. Если данный метод является статическим, то необходимо в атрибуте **class** указать имя класса, содержащего данный метод, а в атрибуте **factory-method** указать имя метода.

Создадим класс Main с точкой входа.

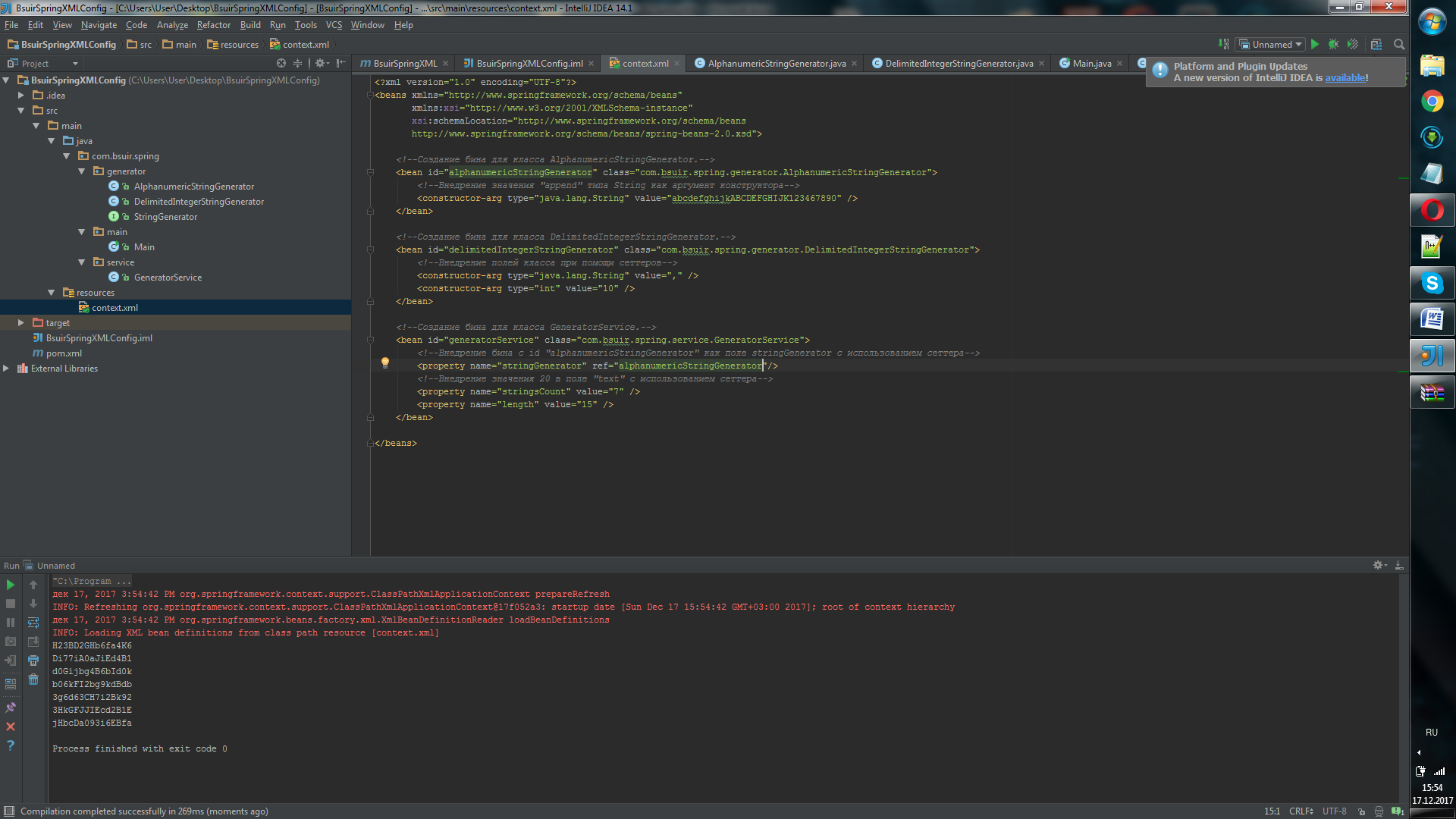
package com.bsuir.spring.main;  
  
import com.bsuir.spring.service.GeneratorService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
import java.util.List;  
  
public class Main {  
  
 private static final String *CONFIGURATION\_PATH* = "context.xml";  
  
 public static void main(String[] args) {  
  
 ApplicationContext context = new ClassPathXmlApplicationContext(*CONFIGURATION\_PATH*);  
   
 GeneratorService service = (GeneratorService) context.getBean("generatorService");  
  
 List<String> generated = service.generateStrings();  
  
 for (String s : generated) {  
 System.*out*.println(s);  
 }  
 }  
}

Результат:

id="delimitedIntegerStringGenerator"

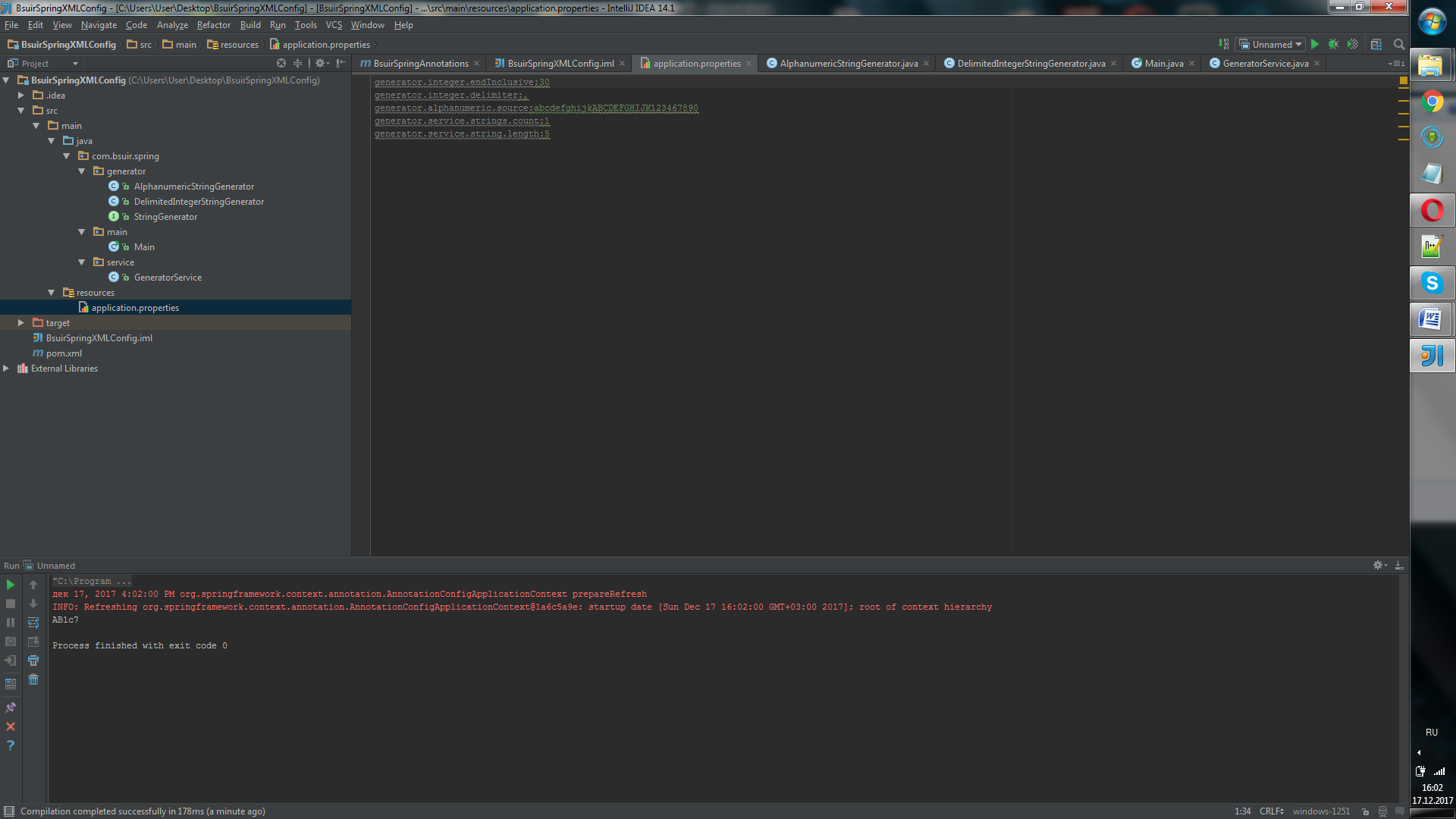


Или id="alphanumericStringGenerator":



Пример 2 (Annotation)

Структура проекта:



По аналогии создадим **pom.xml**.

<?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">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.bsuir.spring</groupId>  
 <artifactId>BsuirSpringAnnotations</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-jar-plugin</artifactId>  
 <configuration>  
 <archive>  
 <manifest>  
 <addClasspath>true</addClasspath>  
 <mainClass>com.bsuir.spring.main.Main</mainClass>  
 <classpathPrefix>dependency-jars/</classpathPrefix>  
 </manifest>  
 </archive>  
 </configuration>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-dependency-plugin</artifactId>  
 <executions>  
 <execution>  
 <id>copy-dependencies</id>  
 <phase>package</phase>  
 <goals>  
 <goal>copy-dependencies</goal>  
 </goals>  
 <configuration>  
 <outputDirectory>  
 ${project.build.directory}/dependency-jars/  
 </outputDirectory>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
 </build>  
  
 <properties>  
 <spring.version>4.3.9.RELEASE</spring.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
 </dependencies>  
  
</project>

Свойства, необходимые для работы программы объявлены в файле **application.properties**. Их применение рассмотрим далее.

generator.integer.endInclusive:30  
generator.integer.delimiter:,  
generator.alphanumeric.source:abcdefghijkABCDEFGHIJK123467890  
generator.service.strings.count:1  
generator.service.string.length:5

Пример реализации интерфейса:

С генерацией строки текста:

package com.bsuir.spring.generator;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
import org.springframework.stereotype.Service;  
  
import java.util.Random;  
  
*/\*\*  
 \* Produces an alphanumeric string.  
 \*/*@Component("alphanumericStringGenerator")  
public class AlphanumericStringGenerator implements StringGenerator {  
  
 private static final Random *RANDOM* = new Random();  
  
 // Default value (after ":") will be used, if there is no value in application.properties  
 @Value("${generator.alphanumeric.source:abcABC123}")  
 private String source;  
  
 @Override  
 public String generate(int count) {  
 if (source == null || source.isEmpty()) {  
 return *EMPTY\_STRING*;  
 }  
 if (count < 0) {  
 throw new IllegalArgumentException("Length of generated string must be positive.");  
 }  
 StringBuilder result = new StringBuilder();  
 for (int index = 0; index < count; index++) {  
 int sourceIndex = *RANDOM*.nextInt(source.length());  
 result.append(source.charAt(sourceIndex));  
 }  
 return result.toString();  
 }  
}

С генерацией строки цифр:

package com.bsuir.spring.generator;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Service;  
  
import java.util.Random;  
  
*/\*\*  
 \* Produces a string full of positive integers.  
 \*/*@Service("delimitedIntegerStringGenerator")  
public class DelimitedIntegerStringGenerator implements StringGenerator {  
  
 private static final Random *RANDOM* = new Random();  
  
 // Default value (after ":") will be used, if there is no value in application.properties  
 @Value("${generator.integer.delimiter:,}")  
 private String delimiter;  
  
 // Default value (after ":") will be used, if there is no value in application.properties  
 @Value("${generator.integer.endInclusive:999}")  
 private int endInclusive;  
  
 @Override  
 public String generate(int count) {  
 if (endInclusive <= 0) {  
 throw new IllegalArgumentException("Property endInclusive must be positive integer.");  
 }  
 if (count < 0) {  
 throw new IllegalArgumentException("Length of generated string must be positive.");  
 }  
 StringBuilder result = new StringBuilder();  
 for (int index = 0; index < count; index++) {  
 result.append(*RANDOM*.nextInt(endInclusive)).append(delimiter);  
 }  
 return result.deleteCharAt(count - 1).toString();  
 }  
}

Для создания бинов с помощью аннотаций необходимо добавить к классу одну из следующих аннотаций: **@Component**, **@Repository**, **@Service**или**@Controller**. При этом аннотация **@Component** является самой общей, а аннотации **@Repository**, **@Service** и**@Controller** различаются семантическим значением. Они используются для уровня персистенции, сервисного уровня и уровня представления соответственно. Но несмотря на различия они имеют одинаковое действие: сообщают контексту, что необходимо создать бин соответствующего класса.

Для рассматриваемого классов будем использовать аннотацию **@Service**.

Для связывания свойств используется аннотация **@Value**. В ней можно указать непосредственно значение, которое необходимо поставить либо используя специальный синтаксис можно подставить значение из файла **\*.property**.

Создадим класс-сервис, осуществляющий прибавление строки к тексту.

package com.bsuir.spring.service;  
  
import com.bsuir.spring.generator.StringGenerator;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Service;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@Service  
public class GeneratorService {  
  
 @Autowired  
 @Qualifier("alphanumericStringGenerator")  
 private StringGenerator stringGenerator;  
  
 @Value("${generator.service.strings.count}")  
 private int stringsCount;  
  
 @Value("${generator.service.string.length}")  
 private int length;  
  
 public List<String> generateStrings() {  
 List<String> strings = new ArrayList<>();  
 for (int count = 0; count < stringsCount; count++) {  
 strings.add(stringGenerator.generate(length));  
 }  
 return strings;  
 }  
}

Внедрение ссылки на бин осуществляется с помощью аннотации **@Autowired**. Данная аннотация осуществляет связывание по типу, т.е. будет внедрен бин, реализующий интерфейс StringGenerator. Если же в контексте зарегистрировано несколько бинов, подходящих под указанный тип, будет сгенерировано исключение **NoUniqueBeanDefinitionException**. Чтобы этого избежать используется аннотация **@Qualifier**, в которой указывается id бина, который необходимо подставить в поле. В данном случае будет подставлен бин с id «alphanumericStringGenerator». Заметим, что для бинов, созданных с помощью аннотаций, id генерируется автоматически как имя класса, записанное с маленькой буквы.

Создадим класс Main с точкой входа.

package com.bsuir.spring.main;  
  
import com.bsuir.spring.service.GeneratorService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
import org.springframework.context.annotation.PropertySource;  
import org.springframework.stereotype.Component;  
  
import java.util.List;  
  
@Component  
@PropertySource("application.properties")  
public class Main {  
  
 private static final String *BASE\_PACKAGE* = "com.bsuir.spring";  
  
 public static void main(String[] args) {  
  
 ApplicationContext context = new AnnotationConfigApplicationContext(*BASE\_PACKAGE*);  
   
 GeneratorService service = context.getBean(GeneratorService.class);  
  
 List<String> generated = service.generateStrings();  
  
 for (String s : generated) {  
 System.*out*.println(s);  
 }  
 }  
}

В данном классе содержится точка входа (метод main). В данном методе конфигурируется контекст, при этом для всех классов, находящихся внутри пакета на любом уровне вложенности и отмеченные аннотациями или **@Component**, **@Repository**, **@Service** и **@Component** будут созданы бины.

Результат работы зависит от указания id бина (alphanumericStringGenerator или delimitedIntegerStringGenerator) в аннотации **@Qualifier.**

Также отметим, что все свойства, используемые для подстановки с помощью аннотации **@Value** задаются в файле application.properties, который подключается здесь же с использованием аннотации **@PropertySource**. Для того чтобы свойства загрузились корректно экземпляр данного класса Main также должен быть бином и регистрироваться в контексте, что и делается с помощью аннотации **@Component**.

Практическая часть

1 часть (Использование XML)

Программа представляет приложение, реализованное на архитектуре Spring.

Варианты заданий:

1. Реализовать приложение позволяющее искать подстроку в строке, а так же переводить ее в верхний регистр.
2. Реализовать калькулятор, имеющей перечень функций (+,-,\*,/).
3. Реализовать приложение заменяющее английский буквы соответствующими русскими буквами по клавиатуре (пример: Q - Й). Учесть регистр.
4. Реализовать программу сортировки строк по алфавиту.
5. Реализовать приложение решения квадратного уравнения.
6. Реализовать приложение конвертации лет в месяцы, недели, дни, часы, секунды.
7. Реализовать вывод числа Фибоначчи с определенного числа.
8. Реализовать калькулятор процентов.
9. Реализовать конвертер десятичных чисел в двоичные.

Контрольные вопросы: