



Universidad
Carlos III de Madrid

TECNOLOGÍAS *DEL* SECTOR FINANCIERO

Práctica 8: Protocolos de mensajería

Andoni Alcelay

MÁSTER UNIVERSITARIO EN TECNOLOGÍAS DEL SECTOR FINANCIERO: FINTECH

1. Introducción

En las siguientes páginas se pueden ver los resultados y resúmenes de los ejercicios realizados en la práctica 8. Esta práctica ha sido dividida en 5 partes, Generar código Java desde un XSD con XJC, Generar instancias para clases con XMLs, generación de XSD con clases java mediante JAXB y generación de código con Velocity.

2. Generación de código Java mediante XJC a partir de un XSD

En este ejercicio se han seguido las instrucciones del documento para ser capaces de generar las clases java necesarias para el resto de la práctica. Con el comando de Java XJC, en la clase XjcLauncher.java, se han añadido los parámetros necesarios para partir desde el xsd practica8.xsd y poder generar dichas clases. El resultado ha sido el siguiente:

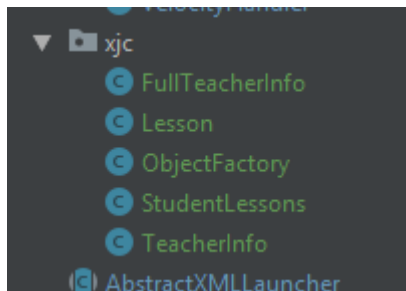


Ilustración 1: Generación de clases con Jaxb

3. Generación de instancias de clases a partir de XMLs y viceversa

En este ejercicio se han generado las instancias de las clases autogeneradas anteriormente con dos ficheros xml, valid.xml y invalid.xml. Para ello se han seguido las instrucciones para modificar la clase JAXBHandler para que pueda interactuar con dichos ficheros. El resultado ha sido el siguiente, en el caso de invalidxml, salta una excepción diciendo el error que ha salido al validar dicho fichero, y por el contrario, al lanzar ValidXML.java, se imprime en pantalla el resultado.

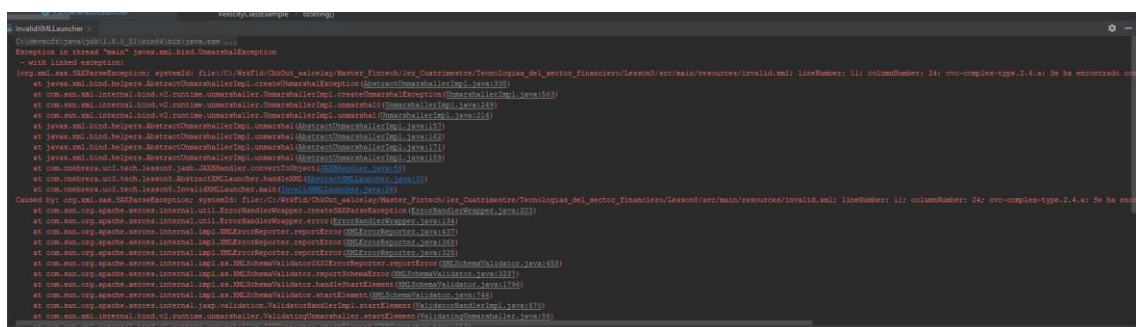
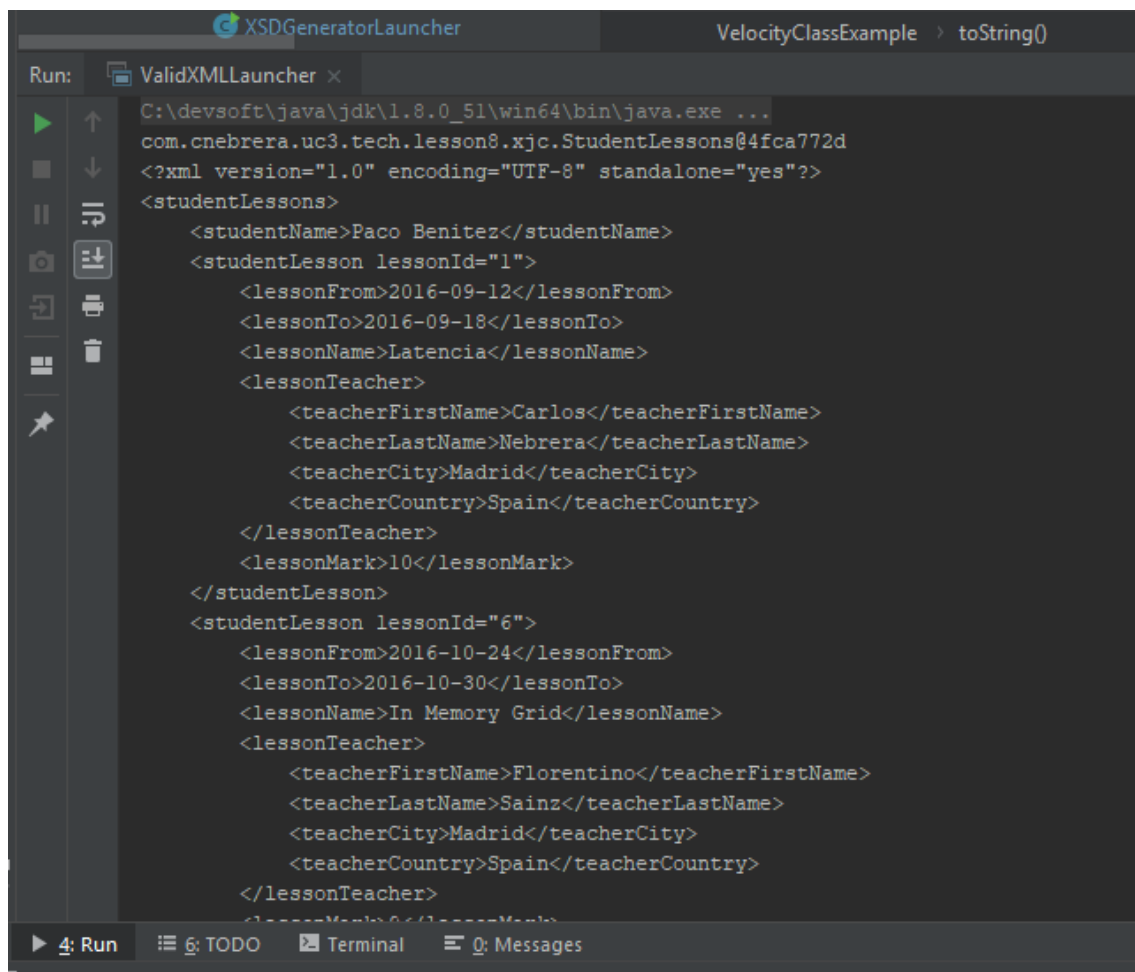


Ilustración 2: Lanzamiento de InvalidXML.java



```

XSDGeneratorLauncher
VelocityClassExample > toString()

Run: ValidXMLLauncher x
C:\devsoft\java\jdk\1.8.0_51\win64\bin\java.exe ...
com.cnebrera.uc3.tech.lesson8.xjc.StudentLessons@4fca772d
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<studentLessons>
  <studentName>Paco Benitez</studentName>
  <studentLesson lessonId="1">
    <lessonFrom>2016-09-12</lessonFrom>
    <lessonTo>2016-09-18</lessonTo>
    <lessonName>Latencia</lessonName>
    <lessonTeacher>
      <teacherFirstName>Carlos</teacherFirstName>
      <teacherLastName>Nebrera</teacherLastName>
      <teacherCity>Madrid</teacherCity>
      <teacherCountry>Spain</teacherCountry>
    </lessonTeacher>
    <lessonMark>10</lessonMark>
  </studentLesson>
  <studentLesson lessonId="6">
    <lessonFrom>2016-10-24</lessonFrom>
    <lessonTo>2016-10-30</lessonTo>
    <lessonName>In Memory Grid</lessonName>
    <lessonTeacher>
      <teacherFirstName>Florentino</teacherFirstName>
      <teacherLastName>Sainz</teacherLastName>
      <teacherCity>Madrid</teacherCity>
      <teacherCountry>Spain</teacherCountry>
    </lessonTeacher>
  </studentLesson>
</studentLessons>

```

Ilustración 3: Lanzamiento de ValidXML.java

4. Generación de XSD a partir de clases anotadas con JAXB

En este ejercicio se genera una plantilla XSD a partir de clases java. Para ello se tocan las clases JAXBHandler y MySchemaOutputResolver. Tras seguir los pasos indicados en esto se ha lanzado el generador de XSD. He tenido un problema ya que tenía el proyecto en el path “1er Cuatrimestre” y este ha dado problemas, ya que identificaba los espacios como %20 y no encontraba el fichero generado. Tras haber corregido esto se ha generado el xsd.

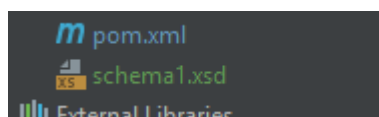


Ilustración 4: Generación de fichero esquema

5. Generación de código con Velocity

Por último, en este ejercicio se ha creado código con velocity, un metalenguaje que con plantillas puede crear código. Para ello se ha cambiado la clase

VelocityGeneratorLauncher.java para ubicar el fichero y generar el código.

En el primer intento se han rellenado los valores de la lista con datos, y este los ha interpretado, en el segundo, en cambio, se han comentado las inserciones de los datos y él automáticamente lanza una excepción.

```
/* Public Constructor
 */
public VelocityClassExample()
{
    this.myListStringValues = new ArrayList<String>() ;

    // Set values from Velocity

    throw new RuntimeException("No string values were injected by Velocity") ;
}

/**
 * @return myListStringValues as string
 */
```

Ilustración 5: Velocity example sin datos

```
/* Public Constructor
 */
public VelocityClassExampleWithData()
{
    this.myListStringValues = new ArrayList<String>() ;

    // Set values from Velocity

    this.myListStringValues.add("Hello") ;
    this.myListStringValues.add("World!") ;
    System.out.println("Velocity injected values!") ;
}

/**
 * @return myListStringValues as string
 */
public String toString() { return myListStringValues.toString() ; }
```

Ilustración 6: Velocity example con datos

Por último se han cambiado las plantillas de velocity, llamada velocityClassExample.vm.