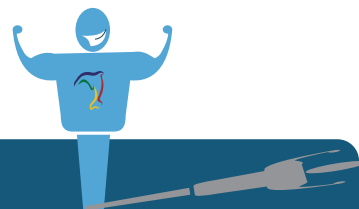


¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

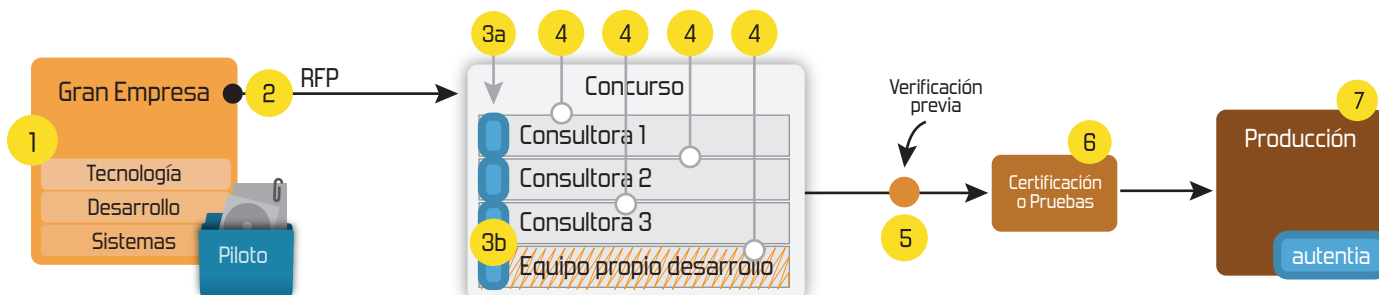
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,
HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

Control de autenticación y
acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)



E-mail:

Contraseña:

Deseo registrarme
He olvidado mis datos de acceso

[Inicio](#) [Quiénes somos](#) [Tutoriales](#) [Formación](#) [Comparador de salarios](#) [Nuestro libro](#) [Charlas](#) [Más](#)



Estás en: [Inicio](#) [Tutoriales](#) [Patrón Intérprete](#)

DESARROLLADO POR:

[Francisco Javier Martínez Páez](#)

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero Técnico en Telecomunicaciones

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Anuncios Google

[Java](#)

[PDF Java API](#)

[Java Log Class](#)

[Manual Java 6](#)

Fecha de publicación del tutorial: 2009-02-26



Share |

[Regístrate para votar](#)

Patrón Intérprete

Lo primero, os dejo el enlace a las: [fuentes de este tutorial](#)

Introducción

En un proyecto en el que estoy participando en este momento (aunque no estoy de técnico, sino de coacher de scrum) se planteó un problema técnico. Se necesitaba crear una forma rápida de definir y ejecutar una especie de fórmula fácilmente modificable. Yo en seguida me acordé del patrón intérprete y de la notación polaca (en lugar de $(A + B) * C$, pues $A B + C *$). Este simple cambio de notación en la expresión permite que sea mucho más sencilla la programación de la ejecución de la operación. Si queréis saber más acerca de todo esto, os dejo un enlace (tirando de éste podéis conocer más acerca del patrón intérprete, de la notación polaca, de cómo pasar de notación infija $(A + B)$ a notación polaca (sufija o prefija)).

Definiendo el lenguaje

Lo primero que haremos será definir el lenguaje de nuestro problema. El ejemplo típico y que mejor se entiende es en el ámbito de operaciones matemáticas:

Símbolo	Operación	Clase Java
+	Suma	SumFunction
-	Resta	SubtractFunction
*	Multipliación	MultFunction
/	División	DivFunction
Cualquier Representación Numérica	Valor numérico	NumberFunction
@	Logaritmo base e	LogeFunction
Cualquier letra [A-Z] [a-z]	Variable	VariableFunction

Yo creo que con esto es suficiente.

Definiendo el intérprete

Necesitaremos también una clase que interprete la expresión y construya la secuencia de funciones según el lenguaje definido. A esta clase la llamaremos (en un alarde de originalidad) InterpreterFunction. Así mismo, para la correcta implementación del patrón, definiremos un interfaz que han de cumplir todas las clases y le denominaremos ExpressionFunction. Además los clientes que utilicen nuestro sistema pueden necesitar pasar información de contexto para la ejecución de la función (por ejemplo las variables a sustituir). Esta información estará dentro la clase ExpressionFunctionContext (por ahora la dejamos en blanco):

```
view plain print ?
package com.autentia.tutoriales.interprete;

import java.math.BigDecimal;

public interface ExpressionFunction {

    public BigDecimal evaluate(ExpressionFunctionContext context);

}

view plain print ?
package com.autentia.tutoriales.interprete;

import java.math.BigDecimal;

public interface ExpressionFunction {

    public BigDecimal evaluate(ExpressionFunctionContext context);

}
```

Empezando con las funciones:

En el código están todas y con sus tests unitarios, os pongo algunas únicamente:

Catálogo de servicios Autentia

Últimas Noticias

- XIV Charla Autentia - ZK - Vídeos y Material
- Hablando de coaching ágil, milagro nocturno y pruebas de vida
- XIII Charla Autentia - AOS y TDD - Vídeos y Material
- Las metodologías ágiles como el catalizador del cambio
- XIV Charla Autentia - ZK

Histórico de NOTICIAS

Últimos Tutoriales

- NIC Bonding, NIC Teaming, Port Trunking, Etherchannel o Ether bonding, con ifenslave en Ubuntu
- CRUD con Spring MVC Portlet (III): Añadiendo validación al formulario
- Tutorial básico de bases de datos en Java mediante JDBC
- Introducción a bases de datos SQL en Java.
- Introducción a bases de datos NoSQL (Not Only SQL)

Últimos Tutoriales del Autor

- MongoDB, primeros pasos
- Spring 3 Java Config Style
- Apache Cassandra, ¿Qué es esto que tanto ruido hace?
- JEE6, haciéndolo fácil.
- Hibernate Search, Bridges, Analizadores y más

Síguenos a través de:



Últimas ofertas de empleo

- 2010-10-11 Comercial - Ventas - SEVILLA.
- 2010-08-30 Otras - Electricidad - BARCELONA.
- 2010-08-24 Otras Sin catalogar - LUGO.

view plain print ?

```
class NumberFunction implements ExpressionFunction {

    private BigDecimal number;

    public NumberFunction(BigDecimal number) {
        this.number = number;
    }

    @Override
    public BigDecimal evaluate(ExpressionFunctionContext context) {
        return this.number;
    }

}
```

view plain print ?

```
class NumberFunction implements ExpressionFunction {

    private BigDecimal number;

    public NumberFunction(BigDecimal number) {
        this.number = number;
    }

    @Override
    public BigDecimal evaluate(ExpressionFunctionContext context) {
        return this.number;
    }

}
```

view plain print ?

```
class DivFunction implements ExpressionFunction {

    private ExpressionFunction dividendo;
    private ExpressionFunction divisor;

    public DivFunction(ExpressionFunction dividendo, ExpressionFunction divisor) {
        this.dividendo = dividendo;
        this.divisor = divisor;
    }

    @Override
    public BigDecimal evaluate(ExpressionFunctionContext context) {
        return dividendo.evaluate(context).divide(divisor.evaluate(context), RoundingMode.HALF_EVEN);
    }

}
```

view plain print ?

```
class DivFunction implements ExpressionFunction {

    private ExpressionFunction dividendo;
    private ExpressionFunction divisor;

    public DivFunction(ExpressionFunction dividendo, ExpressionFunction divisor) {
        this.dividendo = dividendo;
        this.divisor = divisor;
    }

    @Override
    public BigDecimal evaluate(ExpressionFunctionContext context) {
        return dividendo.evaluate(context).divide(divisor.evaluate(context), RoundingMode.HALF_EVEN);
    }

}
```

view plain print ?

```
class LogeFunction implements ExpressionFunction {

    private ExpressionFunction valor;

    public LogeFunction(ExpressionFunction valor) {
        this.valor = valor;
    }

    @Override
    public BigDecimal evaluate(ExpressionFunctionContext context) {
        return new BigDecimal(Math.log(valor.evaluate(context).floatValue()));
    }

}
```

view plain print ?

```
class LogeFunction implements ExpressionFunction {

    private ExpressionFunction valor;

    public LogeFunction(ExpressionFunction valor) {
        this.valor = valor;
    }

    @Override
    public BigDecimal evaluate(ExpressionFunctionContext context) {
        return new BigDecimal(Math.log(valor.evaluate(context).floatValue()));
    }

}
```

2010-06-25

 T. Información - Analista /
Programador - BARCELONA.

view plain print ?

```
public class VariableFunction implements ExpressionFunction {

    private String variableName;

    public VariableFunction(String variableName) {
        this.variableName = variableName;
    }

    @Override
    public BigDecimal evaluate(ExpressionFunctionContext context) {
        return new NumberFunction(getValorFromContext(context)).evaluate(context);
    }

    private BigDecimal getValorFromContext(ExpressionFunctionContext context) {
        return context.getNumberFromContextByVariableName(this.variableName);
    }

}
```

view plain print ?

```
public class VariableFunction implements ExpressionFunction {

    private String variableName;

    public VariableFunction(String variableName) {
        this.variableName = variableName;
    }

    @Override
    public BigDecimal evaluate(ExpressionFunctionContext context) {
        return new NumberFunction(getValorFromContext(context)).evaluate(context);
    }

    private BigDecimal getValorFromContext(ExpressionFunctionContext context) {
        return context.getNumberFromContextByVariableName(this.variableName);
    }

}
```

Para implementar esta clase he necesitado completar la clase ExpressionFunctionContext:

view plain print ?

```
public class ExpressionFunctionContext {

    private Map<String,BigDecimal> mapaVariablesNumeros = null;

    private ExpressionFunctionContext() {
        mapaVariablesNumeros = new HashMap<String, BigDecimal>(3);
    }

    public static ExpressionFunctionContext createExpressionContext(){
        return new ExpressionFunctionContext();
    }

    public ExpressionFunctionContext addVariable(String variableName, BigDecimal valor) {
        mapaVariablesNumeros.put(variableName, valor);
        return this;
    }

    BigDecimal getNumberFromContextByVariableName(String variableName) {
        if(mapaVariablesNumeros==null) {
            throw new ExpressionFunctionException("el contexto es nulo");
        }

        BigDecimal valor = mapaVariablesNumeros.get(variableName);

        if(valor==null){
            throw new ExpressionFunctionException("No se ha encontrado ninguna variable en el contexto con el nombre:"
        }

        return valor;
    }

}
```

view plain print ?

```
public class ExpressionFunctionContext {

    private Map<String,BigDecimal> mapaVariablesNumeros = null;

    private ExpressionFunctionContext() {
        mapaVariablesNumeros = new HashMap<String, BigDecimal>(3);
    }

    public static ExpressionFunctionContext createExpressionContext(){
        return new ExpressionFunctionContext();
    }

    public ExpressionFunctionContext addVariable(String variableName, BigDecimal valor) {
        mapaVariablesNumeros.put(variableName, valor);
        return this;
    }

    BigDecimal getNumberFromContextByVariableName(String variableName) {
        if (mapaVariablesNumeros==null) {
            throw new ExpressionFunctionException("el contexto es nulo");
        }

        BigDecimal valor = mapaVariablesNumeros.get(variableName);

        if(valor==null){
            throw new ExpressionFunctionException("No se ha encontrado ninguna variable en el contexto con el nombre:"
        }

        return valor;
    }
}
```

Para que entendáis cómo se usan, qué mejor que poner los tests de cada una de ellas:

view plain print ?

```
public class NumberFunctionTest {

    @Test
    public void probando_number_function_test() {
        NumberFunction numberFunction = new NumberFunction(new BigDecimal(8.7));
        Assert.assertEquals(new BigDecimal(8.7), numberFunction.evaluate(null));
    }

}

public class DivFunctionTest {

    @Test
    public void probando_div_function_con_numeros_test() {

        DivFunction division = new DivFunction(new NumberFunction(new BigDecimal(6.4f)), new NumberFunction(new BigDec

        BigDecimal expected = new BigDecimal(6.4f).divide(new BigDecimal(3.0f),RoundingMode.HALF_EVEN);
        Assert.assertEquals(expected, division.evaluate(null));

    }

    @Test
    public void probando_div_function_con_numeros_y_variables_test() {

        DivFunction division = new DivFunction(new NumberFunction(new BigDecimal(8.7f)), new VariableFunction("X"));

        BigDecimal expected = new BigDecimal(8.7f).divide(new BigDecimal(6.3f),RoundingMode.HALF_EVEN);
        Assert.assertEquals(expected, division.evaluate(ExpressionFunctionContext.createExpressionContext().addVariabl

    }

}

public class VariableFunctionTest {

    @Test
    public void probando_variable_function_con_variable_en_el_contexto_test() {

        ExpressionFunctionContext context = ExpressionFunctionContext.createExpressionContext().addVariable("A", new E

        VariableFunction function = new VariableFunction("A");

        Assert.assertEquals(new BigDecimal(7.90), function.evaluate(context));

    }

    @Test(expected=ExpressionFunctionException.class)
    public void probando_variable_function_sin_variable_en_el_contexto_test() {

        ExpressionFunctionContext context = ExpressionFunctionContext.createExpressionContext();

        VariableFunction function = new VariableFunction("A");

        Assert.assertEquals(new BigDecimal(7.90), function.evaluate(context));

    }

}
```

view plain print ?

```
public
class NumberFunctionTest {

    @Test
    public void probando_number_function_test() {
        NumberFunction numberFunction = new NumberFunction(new
BigDecimal(8.7));
        Assert.assertEquals(new BigDecimal(8.7),
numberFunction.evaluate(null));
    }

}

public class DivFunctionTest {

    @Test
    public void probando_div_function_con_numeros_test() {

        DivFunction division = new DivFunction(new NumberFunction(new
BigDecimal(6.4f)), new NumberFunction(new BigDecimal(3.0f)));

        BigDecimal expected = new BigDecimal(6.4f).divide(new
BigDecimal(3.0f), RoundingMode.HALF_EVEN);
        Assert.assertEquals(expected, division.evaluate(null));

    }

    @Test
    public void probando_div_function_con_numeros_y_variables_test() {

        DivFunction division = new DivFunction(new NumberFunction(new
BigDecimal(8.7f)), new VariableFunction("X"));

        BigDecimal expected = new BigDecimal(8.7f).divide(new
BigDecimal(6.3f), RoundingMode.HALF_EVEN);
        Assert.assertEquals(expected,
division.evaluate(ExpressionFunctionContext.createExpressionContext().addVariable("X",
new BigDecimal(6.3f))));

    }

}

public class VariableFunctionTest {

    @Test
    public void
probando_variable_function_con_variable_en_el_contexto_test() {

        ExpressionFunctionContext context =
ExpressionFunctionContext.createExpressionContext().addVariable("A", new
BigDecimal(7.90));

        VariableFunction function = new VariableFunction("A");

        Assert.assertEquals(new BigDecimal(7.90), function.evaluate(context));

    }

    @Test(expected=ExpressionFunctionException.class)
    public void
probando_variable_function_sin_variable_en_el_contexto_test() {

        ExpressionFunctionContext context =
ExpressionFunctionContext.createExpressionContext();

        VariableFunction function = new VariableFunction("A");

        Assert.assertEquals(new BigDecimal(7.90), function.evaluate(context));

    }

}
```

Ahora vamos a la clase que hace que esto se empiece a entender. Para poder hacer uso de la notación polaca usaremos una pila (Stack) parecido a como lo hacen las calculadoras HP (por lo menos en mis tiempos de universidad) . Defino la clase como un servicio. De esta manera la pila de ejecuciones ya estará creada. Además, usaremos un fichero de propiedades para definir la expresión que ejecuta. Os la enseño:

view plain print ?

```
@Service
public class InterpreterFunction implements ExpressionFunction {

    private ExpressionFunction expressionsTree;

    private Stack<ExpressionFunction> expressionsStack;

    private String actualExpression;

    public String getActualExpression() {
        return actualExpression;
    }

    @Autowired
    public InterpreterFunction(@Value("#{props['interpreter.expression']}")String expression) {
        this.actualExpression = expression;
        expressionsStack = new Stack<ExpressionFunction>();

        for (String token : expression.split(" ")) {
            ExpressionFunction subFunction = createTheRightFunction(token);
            expressionsStack.push(subFunction);
        }
        expressionsTree = expressionsStack.pop();
    }

    //TODO considerar extraer esto a una Factoría de familias de funciones. Para una sesión de refactor:
    private ExpressionFunction createTheRightFunction(String token) {

        ExpressionFunction funcion = null;

        if (NumberUtils.isNumber(token)) {
            funcion = new NumberFunction(new BigDecimal(token));
        } else if (StringUtils.isAlpha(token)) {
            funcion = new VariableFunction(token);
        } else if ("+".equals(token)) {
            funcion = new SumFunction(expressionsStack.pop(), expressionsStack.pop());
        } else if ("-".equals(token)) {
            ExpressionFunction right = expressionsStack.pop();
            ExpressionFunction left = expressionsStack.pop();
            funcion = new SubtractFunction(left, right);
        } else if ("*".equals(token)) {
            funcion = new MultFunction(expressionsStack.pop(), expressionsStack.pop());
        } else if ("/".equals(token)) {
            ExpressionFunction right = expressionsStack.pop();
            ExpressionFunction left = expressionsStack.pop();
            funcion = new DivFunction(left, right);
        } else if ("%".equals(token)) {
            funcion = new LogeFunction(expressionsStack.pop());
        } else {
            throw new ExpressionFunctionException("regla no reconocida: " + token);
        }

        return funcion;
    }

    public BigDecimal evaluate(ExpressionFunctionContext context) {
        return expressionsTree.evaluate(context);
    }
}
```

view plain print ?

```
@Service
public class InterpreterFunction implements ExpressionFunction {

    private ExpressionFunction expressionsTree;

    private Stack<ExpressionFunction> expressionsStack;

    private String actualExpression;

    public String getActualExpression() {
        return actualExpression;
    }

    @Autowired
    public InterpreterFunction(@Value("#{props['interpreter.expression']}")String expression) {
        this.actualExpression = expression;
        expressionsStack = new Stack<ExpressionFunction>();

        for (String token : expression.split(" ")) {
            ExpressionFunction subFunction = createTheRightFunction(token);
            expressionsStack.push(subFunction);
        }
        expressionsTree = expressionsStack.pop();
    }

    //TODO considerar extraer esto a una Factoría de familias de funciones. Para una sesión de refactor:
    private ExpressionFunction createTheRightFunction(String token) {

        ExpressionFunction funcion = null;

        if (NumberUtils.isNumber(token)) {
            funcion = new NumberFunction(new BigDecimal(token));
        } else if (StringUtils.isAlpha(token)) {
            funcion = new VariableFunction(token);
        } else if ("+".equals(token)) {
            funcion = new SumFunction(expressionsStack.pop(), expressionsStack.pop());
        } else if ("-".equals(token)) {
            ExpressionFunction right = expressionsStack.pop();
            ExpressionFunction left = expressionsStack.pop();
            funcion = new SubtractFunction(left, right);
        } else if ("*".equals(token)) {
            funcion = new MultFunction(expressionsStack.pop(), expressionsStack.pop());
        } else if ("/".equals(token)) {
            ExpressionFunction right = expressionsStack.pop();
            ExpressionFunction left = expressionsStack.pop();
            funcion = new DivFunction(left, right);
        } else if ("%".equals(token)) {
            funcion = new LogeFunction(expressionsStack.pop());
        } else {
            throw new ExpressionFunctionException("regla no reconocida: " + token);
        }

        return funcion;
    }

    public BigDecimal evaluate(ExpressionFunctionContext context) {
        return expressionsTree.evaluate(context);
    }
}
```

Y ahora una par de clases de tests. Una para probar que funciona correctamente sin contar con Spring. Usamos un Runner de JUnit llamado Parameterized que permite pasar parámetros al test. Así conseguimos lanzar 6 tests en uno con diferentes parámetros.

view plain print ?

```
@RunWith(value = Parameterized.class)
public class InterpreterFunctionTest {

    private String expresion;
    private BigDecimal resultado;

    static ExpressionFunctionContext contexto = null;

    @Parameters
    public static Collection<Object[]> getTestParameters() {
        contexto = ExpressionFunctionContext.createExpressionContext().addVariable("X", new BigDecimal(8)).addVariable("Y", new BigDecimal(9));
        return Arrays.asList(new Object[][] {
            { "7 8 + 50 -", new BigDecimal("-35") },
            { "0 0 +", new BigDecimal("0") },
            { "6.54 99.76 + 45 50 - - X +", new BigDecimal("119.30") },
            { "6.54 99.76 * 45 50 - - X +", new BigDecimal("665.43") },
            { "6.54 99.76 * 45 50 / Y * - @", new BigDecimal("6.46") },
            { "X Y * 7 +", new BigDecimal("63") }
        });
    }

    public InterpreterFunctionTest(String regla, BigDecimal resultado) {
        this.expresion = regla;
        this.resultado = resultado;
    }

    @Test
    public void testingSomeOperationsExpression() {
        InterpreterFunction evaluator = new InterpreterFunction(expresion);
        assertEquals(resultado.doubleValue(), evaluator.evaluate(contexto).doubleValue(), 0.1);
    }
}
```

view plain print ?

```
@RunWith(value
    = Parameterized.class)
public class InterpreterFunctionTest {

    private String expresion;
    private BigDecimal resultado;

    static ExpressionFunctionContext contexto = null;

    @Parameters
    public static Collection<Object[]> getTestParameters() {
        contexto =
        ExpressionFunctionContext.createExpressionContext().addVariable("X", new
        BigDecimal(8)).addVariable("Y", new BigDecimal(7));

        return Arrays.asList(new Object [][] {

            { "7 8 + 50 -",new BigDecimal("-35")},
            { "0 0 +", new BigDecimal("0")},
            { "6.54 99.76 + 45 50 - - X +",new BigDecimal("119.30")},
            { "6.54 99.76 * 45 50 - - X +",new BigDecimal("665.43")},
            { "6.54 99.76 * 45 50 / Y * - @",new BigDecimal("6.46")},
            { "X Y * 7 +",new BigDecimal("63")}

        });

    }

    public InterpreterFunctionTest(String regla, BigDecimal resultado) {
        this.expresion = regla;
        this.resultado = resultado;
    }

    @Test
    public void testingSomeOperationsExpression() {
        InterpreterFunction evaluator = new InterpreterFunction(expresion);
        assertEquals(resultado.doubleValue(),
            evaluator.evaluate(contexto).doubleValue(),0.1);
    }

}
```

Ahora otro test para comprobar que Spring me lo crea correctamente y me inyecta la expresión:

view plain print ?

```
@ContextConfiguration(locations={"classpath:applicationContext-test.xml"})
@RunWith(SpringJUnit4ClassRunner.class)
public class InterpreterFunctionSpringTest {

    @Autowired
    InterpreterFunction interpreter;

    @Value("#{props['interpreter.expression']}")
    String expression1;

    @Value("#{props['interpreter.result']}")
    String result;

    @Test
    public void inyeccion_expression_correcta_Test() {
        Assert.assertEquals(expression1,interpreter.getActualExpression());
    }

    @Test
    public void evaluacion_correcta_suma_simple_Test() {
        BigDecimal expected = new BigDecimal(result);

        Assert.assertEquals(expected,interpreter.evaluate(null));
    }

}
```

view plain print ?

```
@ContextConfiguration(locations={"classpath:applicationContext-test.xml"})
@RunWith(SpringJUnit4ClassRunner.class)
public class InterpreterFunctionSpringTest {

    @Autowired
    InterpreterFunction interpreter;

    @Value("#{props['interpreter.expression']}")
    String expression1;

    @Value("#{props['interpreter.result']}")
    String result;

    @Test
    public void inyeccion_expression_correcta_Test() {
        Assert.assertEquals(expression1,interpreter.getActualExpression());
    }

    @Test
    public void evaluacion_correcta_suma_simple_Test() {
        BigDecimal expected = new BigDecimal(result);

        Assert.assertEquals(expected,interpreter.evaluate(null));
    }

}
```

El fichero del contexto de Spring:

view plain print ?

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-3.0.xsd">

  <context:component-scan base-package="com.autentia.tutoriales.interprete" />

  <util:properties id="props" location="classpath:interpreterTest.properties"/>

</beans>
```

view plain print ?

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-3.0.xsd">

  <context:component-scan base-package="com.autentia.tutoriales.interprete" />

  <util:properties id="props" location="classpath:interpreterTest.properties"/>

</beans>
```

El fichero de propiedades es muy simple:

view plain print ?

```
interpreter.expression = 6.54 99.76 + 45 50 - - 8 +
interpreter.result = 119.30
```

view plain print ?

```
interpreter.expression = 6.54 99.76 + 45 50 - - 8 +
interpreter.result = 119.30
```

Bueno, pues nos hemos hecho un mini motor de reglas con 11 ó 12 clases (aunque en este caso son operaciones matemáticas), ahora sólo queda buscarle sentido en tu negocio.

Reconocimientos

Gracias a Iván Zaera que hace ya muchos años me dejó patidifuso cuando me contó lo de la Notación Polaca (los de Teleco no solemos saber de estas cosas y las aprendemos gracias a los informáticos de verdad).

Anímate y coméntanos lo que pienses sobre este **TUTORIAL**:

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario

(Sólo para usuarios registrados)

» **Regístrate** y accede a esta y otras ventajas «

COMENTARIOS



Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5