

[Home](#)[Newsletter](#)[Java Specialist Club](#)[Java Training](#)[Conference Venue Hire](#)[Java Resources](#)[Contact](#)

The Java Specialists' Newsletter

➤ Issue 097 ➤ 2004-10-17 ➤ Category: **Language** ➤ Java version: Sun JDK 1.5.0-b64

0

[Subscribe Free](#)[RSS Feed](#)[Print](#)

Mapping Objects to XML Files using Java 5 Annotations

by Amotz Anner

Welcome to the 97th edition of **The Java(tm) Specialists' Newsletter**. We had a beautiful day down here in Cape Town, South Africa. This morning I was concerned because it was cloudy, but fortunately it cleared up for our braai. In South Africa, a popular pass-time is to make a fire with wood, then once this has burnt down, to grill some lamb chops, sausage or chicken on the coals. We only braai once the coals are completely burnt, to avoid having flames leap up and burn the meat. The coals have to be so hot that you cannot hold your hand at the level of the meat for more than eight seconds. Some wood holds its heat longer, and you must make sure that the flavour of the meat is not adversely affected by the wood you choose. The braai is more for the social interaction than for the food and so it is common to start the fire only after the guests have arrived. Another interesting tradition is to invite guests for a braai, but ask them to bring their own meat and drinks, and maybe a salad. We call that a bring-and-braai. The company is very important, and although we sometimes braai on our own, we usually prefer to have lots of happy friendly faces around. For example, today we were 17!

So, if you visit South Africa, and tell me when you will be in Cape Town, you will probably be invited to a "braai". We will probably ask you to bring some of the food, and when you arrive, I will still be chopping the wood, but besides that, it is a lot of fun, and you should accept the invitation :)

I find it amazing how quickly some people pick up a new technology. Amotz Anner sent me an example of how you can use annotations to write some very interesting code. Then Amotz did me a huge favour and wrote this newsletter. Thank you very very much.

Amotz is the **Founder of X.M.L. Systems Ltd** located on:
46, Jerusalem St. Flat 9B
Kfar-Saba 44369
Israel
Cell: +972 (54) 686-0707

That was not the only interesting communication with Amotz. He discovered that in the JDK 1.5.0, if you compile code that uses the ternary if-else operator, with one side of the results **null** then it can deliver wrong results if you immediately assign this to a **final** local variable. This compiler bug has been fixed and will be available in the next release. For more details, please look at our posts on [JavaLobby](#) and [TheServerSide](#).

This bug does concern me. With most Java bugs that I have seen, upgrading to a new JRE solves the problem. With this bug, you need to upgrade AND recompile all your sources. It is easy for this bug to make it into your production code library, and unless you have a very good test procedure, you might never pick it up. So, be warned!

Enough from me, over to Amotz...

Upcoming Java Master Courses:

Duesseldorf, Germany, Aug 22
Chania, Crete, Sep 6
Cape Town, South Africa, Sep 12

In-house courses if these dates or locations do not suit you. Note that the course in Crete may also be attended remotely via webinar.

Mapping Objects to XML Files using Java 5 Annotations

The following is an example of using the new annotation capability of Java 5.0 to extend the expressiveness of Java.

My code relies on XML to declare all sort of components, and then have Java classes construct themselves from those declarations. It is NOT a persistence framework, for the following reason:

I consider the XML declaration to be primary, and the Java class to be secondary, a mere tool to realize the declaratory intent of the XML document. In contrast, in a persistence framework, the Java class is primary and the XML document is just a vehicle used to contain persistence data, and has no standing in and of itself.

So my requirement is that the Java class adapts itself to the XML declaration rather than the other way around. I also did not want to use any external IDL-type definitions to match Java classes to XML.

Newsletter Links

- [Book Review](#)
- [Concurrency](#)
- [Exceptions](#)
- [GUI](#)
- [Inspirational](#)
- [Language](#)
- [Performance](#)
- [Software Engineering](#)
- [Tips and Tricks](#)

Java Courses

- Java Master
- Java Foundation
- Java 5 Tiger
- Design Patterns

➤ [find out more](#)

What is the Java Specialists Club?

Venue for Hire

Looking for a super conference room for your next company event?



Crete is your perfect destination.

[Click here for more details](#)

Prior to annotations, I was severely limited in my choices. What I could, and did do, was to use reflection to look for all public fields of a class whose names match those of an XML attribute in the appropriate declaration, and initialize those fields from the attribute value. This was a fragile solution, since there was no clear indication as to the special status of the names of those fields, and all too often I broke my code by changing field names, thus breaking the connection to the XML declaration.

Then came annotations and solved all my problems in one fell swoop. With them I can:

1. Clearly indicate which fields are initialized from an XML declaration.
2. Dissolve the field name - attribute name bond.
3. Extend usage from just XML attributes to XML elements as well.
4. Supply centrally/locally defined default values.

So how is that magic achieved? In four easy steps, of course.

Step 1: Annotation is defined

First, an annotation is defined, in the same way as an interface would be, as follows:

```
import java.lang.annotation.*;

/**
 * Make annotation available at run time and only allow fields to
 * be modified by this annotation.
 *
 * We have 3 properties, defaults for all
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface FromXml {
    /**
     * Normally, the field's value is taken from an attribute with
     * an identical name. XPath can be used to specify a different
     * source, breaking the name linkage
     */
    String xpath() default "";
    /**
     * A default value to be used if field source is not found in
     * the XML (we call it "dflt" since "default" is reserved)
     */
    String dflt() default "";
    /**
     * Flag to trim fetched data.
     */
    boolean trim() default true;
}
```

Step 2: Fields are Annotated

Second, fields have to be annotated. Syntactically, annotations are modifiers, so the result looks like:

```
// This is equivalent to old usage.
@FromXml public int a;
// This is the new usage.
@FromXml(xpath = "a/b/c", dflt = "blue") public String color;
```

Step 3: Call initializer

The third easy step is to call the initializer with a Class and Element references:

```
xmlConstructor.constructFromXml(this, elem, false);
```

A more complete example is our ComponentSlider class that manages a slider that can be configured using XML and annotations. You need the **dom4j** jar file to get this to compile.

```
import org.dom4j.Element;
import javax.swing.*;

public class ComponentSlider {
    @FromXml private boolean inverted = false;
    @FromXml private int min = Integer.MIN_VALUE;
    @FromXml private int max = Integer.MIN_VALUE;
    @FromXml private int minorTickInterval = Integer.MIN_VALUE;
    @FromXml private int majorTickInterval = Integer.MIN_VALUE;
    @FromXml private boolean snapToTick = false;

    public ComponentSlider(JSlider slider, Element def) {
        XmlConstructor.constructFromXml(this, def);
        slider.setMinimum(min);
        slider.setMaximum(max);
        slider.setInverted(inverted);
        if (minorTickInterval != Integer.MIN_VALUE) {
            slider.setMinorTickSpacing(minorTickInterval);
            slider.setPaintTicks(true);
            slider.setSnapToTicks(snapToTick);
        }
        if (majorTickInterval != Integer.MIN_VALUE) {
            slider.setMajorTickSpacing(majorTickInterval);
            slider.setPaintTicks(true);
            slider.setPaintLabels(true);
            slider.setSnapToTicks(snapToTick);
        }
    }
}
```

Step 4: Supply constructFromXml method (once)

And finally, but just once, the above method has to be supplied. It looks like:

```
import org.dom4j.*;
import java.lang.reflect.Field;

public class XmlConstructor {
    public static void constructFromXml(Object obj, Element elem) {
        constructFromXml(obj, elem, false);
    }

    /**
     * Set object's fields from values of XML declarations, using
     * "@FromXml" annotation
     *
     * Scans all fields in an object for a annotated elements, and
     * initialize them, according to the fields type.
     *
     * @param usesSuper If super class is to be processed
     * @param o The object to scan for fields
     * @param element The element whose attributes are the data
     *                sources.
     */
    public static void constructFromXml(Object o, Element element,
                                       boolean usesSuper) {
        if (element == null) {
            return;
        }
        Class aClass = getAppropriateClass(o, usesSuper);

        for (Field field : aClass.getDeclaredFields()) {
            FromXml fromXml = field.getAnnotation(FromXml.class);
            if (fromXml != null) {
                field.setAccessible(true);
                handleAnnotatedField(fromXml, element, field, o);
            }
        }
    }

    private static void handleAnnotatedField(FromXml fromXml,
                                             Element element,
                                             Field field, Object o) {
        String value = getValue(fromXml.xpath(), element, field, fromXml);
        if (!isEmpty(value)) {
            if (fromXml.trim()) {
                value = value.trim();
            }
            setField(field, o, value);
        }
    }

    private static String getValue(String xpath, Element element,
                                   Field field, FromXml fromXml) {
        String value = null;
        if (!isEmpty(xpath)) {
            Node node = element.selectSingleNode(xpath);
            if (node != null) {
                value = node.getText();
            }
        } else {
            // If no xpath, use name
            // Get value of matching attribute
            value = element.attributeValue(field.getName());
        }

        if (value == null) {
            //Use default
            value = fromXml.dflt();
        }
        return value;
    }

    private static void setField(Field field, Object o, String value) {
        Class type = field.getType();
        try {
            // Now initialize field according to type
            if (type.equals(int.class)) {
                field.setInt(o, asHexInt(value));
            } else if (type.equals(String.class)) {
                field.set(o, value.intern());
            } else if (type.equals(double.class)) {
                field.setDouble(o, Double.parseDouble(value));
            } else if (type.equals(boolean.class)) {
                field.setBoolean(o, asBoolean(value));
            } else if (type.equals(char.class)) {
                field.setChar(o, value.charAt(0));
            }
        } catch (IllegalAccessException ex) {
            // this should not happen, since we are setting the access
            // to true
            throw new RuntimeException(ex);
        }
    }

    private static Class getAppropriateClass(Object o, boolean usesSuper) {
        Class aClass = o.getClass();
        if (usesSuper) {
            aClass = aClass.getSuperclass();
        }
        return aClass;
    }

    private static boolean isEmpty(String test) {
        return test == null || test.length() == 0;
    }
}
```

```

/**
 * Use hex conversion if string starts with "0x", else decimal
 * conversion.
 */
private static int asHexInt(String value) {
    if (value.toLowerCase().startsWith("0x")) {
        return Integer.parseInt(value.substring(2), 16);
    }
    return Integer.parseInt(value);
}

private static boolean asBoolean(String option) {
    if (!isEmpty(option)) {
        String opt = option.toLowerCase();
        return "yes".equals(opt) || "on".equals(opt)
            || "true".equals(opt) || "1".equals(opt);
    }
    return false;
}
}

```

Very simple, really.

Next, we create an XML file that contains the attributes for the Slider class:

```

<?xml version="1.0" encoding="UTF-8"?>
<Slider xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:noNamespaceSchemaLocation='file:/C:/XML%20SYS/XML%20Sys/XSP/Components.xsd'
  min="20" max="180" minorTickInterval="2" majorTickInterval="10">
  <xmlPath>checkup/weight</xmlPath>
</Slider>

```

And an example class that uses the ComponentSlider:

```

import org.dom4j.*;
import org.dom4j.io.SAXReader;
import javax.swing.*;
import java.io.*;

public class AnnotationDemo extends JFrame {
    private JSlider slider = new JSlider();

    public AnnotationDemo(Element sliderDef) {
        new ComponentSlider(slider, sliderDef);
        // HK: did you notice that you don't have to say:
        // getContentPane().add(...) in JDK 1.5 anymore?
        add(slider);
    }

    private static Element loadSliderXMLFile(String filename)
        throws FileNotFoundException, DocumentException {
        // The slider definition as XML
        Element sliderDef = null;
        // A reusable SAX parser
        SAXReader xmlReader = new SAXReader();
        xmlReader.setIgnoreComments(true);
        xmlReader.setMergeAdjacentText(true);
        xmlReader.setStripWhitespaceText(true);
        File file = new File(filename);
        if (file.exists() && file.canRead()) {
            Document doc = xmlReader.read(new FileInputStream(file));
            sliderDef = doc.getRootElement();
        }
        if (sliderDef == null) {
            throw new IllegalArgumentException(
                "Could not find XML declaration");
        }
        return sliderDef;
    }

    public static void main(String args[]) throws Exception {
        Element sliderDef = loadSliderXMLFile(args[0]);
        AnnotationDemo frame = new AnnotationDemo(sliderDef);
        frame.setSize(500, 100);
        frame.setDefaultCloseOperation(EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null); // center the frame
        frame.setVisible(true);
    }
}

```

I have not done it yet, but I can now initialize multi-dimensional arrays, lists, collections or what-not by straightforward extensions of the above code. Obviously, it made no sense before annotations, as XML attributes appear at most once. But with elements, there is no such limitation, and there is nothing to prevent database queries and/or other trickery.

Left as an exercise to the reader, is another annotation which would be used to associate "slider" directly with "weightSlider.xml", making annotation usage consistent, well documented and safe.

Have fun.

Amotz

P.S. I (Heinz) refactored Amotz's code a bit, so if you find bugs or do not like the style of the code, blame me :)

➤ [Language Articles](#) ➤ [Related Java Course](#) ➤ [Discuss at The Java Specialist Club](#)

© 2010 Heinz Kabutz - All Rights
Reserved

[Sitemap](#)

[seo web design](#) Catch22 Marketing

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. JavaSpecialists.eu is not connected to Oracle, Inc. and is not sponsored by Oracle, Inc.