Programmieren II

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2010

JAXB

Javadoc

JAXB

JAXB ist die *Java API for XML Binding*. Diese Programmschnittstelle erlaubt,

- Daten aus einer XML-Schema-Instanz heraus automatisch an Java-Klassen zu binden, und
- diese Java-Klassen aus einem XML-Schema heraus zu generieren.

JAXB ermöglicht auch den umgekehrten Weg, d. h. das Erstellen eines Schemas aus Java-Klassen (mit speziellen Annotationen).

Workflow

- Zunächst wird ein XML-Schema für die Daten erstellt (hier: `kontakt.xsd`)
- Dann wird mit dem Compiler `xjc` die entsprechende Klassenhierarchie erzeugt, z. B.
  ```
  xtc -d src -p kontakt kontakt.xsd
  ```
  erzeugt Java-Klassen zum Schema innerhalb des Paketes kontakt, das sich im Verzeichnis `src` befindet.
- Dort entstehen die Klassen RootType und PersonType mit entsprechenden Attributen, Gettern und Settern, die die komplexten Typen im Schema abbilden.
- Weiterhin wird eine Klasse ObjectFactory erzeugt, mit der neue Instanzen erzeugt werden können, die dann in XML abgebildet werden.

Schaubild
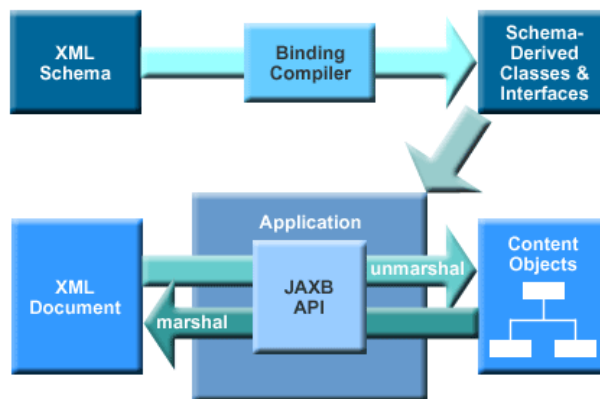
Beispiel

```java
import java.io.*;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.Unmarshaller;
import kontakt.*;

public class JAXBParser {
    public static void main(String[] args) {
        try {
            JAXBContext jc = JAXBContext.newInstance("kontakt");
            Unmarshaller u = jc.createUnmarshaller();
            JAXBElement<?> kElement =
                (JAXBElement<?>) u.unmarshal(
                        new FileInputStream("personen.xml"));
            RootType kontakte = (RootType) kElement.getValue();
```

Beispiel (Forts.)

```java
        for (PersonType person : kontakte.getPerson()) {
            System.out.println(person.getNachname());
            System.out.println(person.getFirma());
            System.out.println(person.isVip());
        }
    } catch (JAXBException e) {
        e.printStackTrace();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
  }
}
```

Javadoc

Javadoc dient der Generierung von API-Dokumentationen:

- Pakete
- Klassen
- Methoden mit
  - Parametern
  - Rückgabewert

anhand von *strukturierten Kommentaren* im Quelltext.

Kommentarformat

Javadoc-Kommentare haben die Form

```
/**
Javadoc-Kommentar
*/
```

Hierbei werden simple HTML-Tags innerhalb der Kommentare und bestimmte mit @ beginnende Tags erkannt.

Tags

Tag	Wo
@author	Klasse, Interface
@version	Klasse, Interface
@param	Konstruktor, Methode
@return	Methode
@exception/@throws	Methode
@see	
@since	
@deprecated	

Beispiel

```
/**
 * Graphics is the abstract base class for all graphics contexts
 * which allow an application to draw onto components realized on
 * various devices or onto off-screen images.
 * A Graphics object encapsulates the state information needed
 * for the various rendering operations that Java supports.  This
 * state information includes:
 * <ul>
 * <li>The Component to draw on
 * <li>A translation origin for rendering and clipping coordinates
 * <li>The current clip
 * <li>The current color
 * <li>The current font
 * <li>The current logical pixel operation function (XOR or Paint)
 * <li>The current XOR alternation color
 *     (see <a href="#setXORMode">setXORMode</a>)
 * </ul>
 * <p>
 * Coordinates are infinitely thin and lie between the pixels of the
 * output device.
 * Operations which draw the outline of a figure operate by traversing
 * along the infinitely thin path with a pixel-sized pen that hangs
```

Beispiel

```
 * down and to the right of the anchor point on the path.
 * Operations which fill a figure operate by filling the interior
 * of the infinitely thin path.
 * Operations which render horizontal text render the ascending
 * portion of the characters entirely above the baseline coordinate.
 * <p>
 * Some important points to consider are that drawing a figure that
 * covers a given rectangle will occupy one extra row of pixels on
 * the right and bottom edges compared to filling a figure that is
 * bounded by that same rectangle.
 * Also, drawing a horizontal line along the same y coordinate as
 * the baseline of a line of text will draw the line entirely below
 * the text except for any descenders.
 * Both of these properties are due to the pen hanging down and to
 * the right from the path that it traverses.
 * <p>
 * All coordinates which appear as arguments to the methods of this
 * Graphics object are considered relative to the translation origin
 * of this Graphics object prior to the invocation of the method.
 * All rendering operations modify only pixels which lie within the
 * area bounded by both the current clip of the graphics context
 * and the extents of the Component used to create the Graphics object.
```

Beispiel

```
 *
 * @author      Sami Shaio
 * @author      Arthur van Hoff
 * @version     %I%, %G%
 * @since       1.0
 */
public abstract class Graphics {

    /**
     * Draws as much of the specified image as is currently available
     * with its northwest corner at the specified coordinate (x, y).
     * This method will return immediately in all cases, even if the
     * entire image has not yet been scaled, dithered and converted
     * for the current output device.
     * <p>
     * If the current output representation is not yet complete then
     * the method will return false and the indicated
     * {@link ImageObserver} object will be notified as the
     * conversion process progresses.
     *
     * @param img       the image to be drawn
     * @param x         the x-coordinate of the northwest corner
```

Beispiel

```
     *               of the destination rectangle in pixels
     * @param y       the y-coordinate of the northwest corner
     *               of the destination rectangle in pixels
     * @param observer  the image observer to be notified as more
     *               of the image is converted.  May be
     *               <code>null</code>
     * @return        <code>true</code> if the image is completely
     *               loaded and was painted successfully;
     *               <code>false</code> otherwise.
     * @see           Image
     * @see           ImageObserver
     * @since         1.0
     */
    public abstract boolean drawImage(Image img, int x, int y,
                    ImageObserver observer);


    /**
     * Dispose of the system resources used by this graphics context.
     * The Graphics context cannot be used after being disposed of.
     * While the finalization process of the garbage collector will
     * also dispose of the same system resources, due to the number
```

Beispiel

```
     * of Graphics objects that can be created in short time frames
     * it is preferable to manually free the associated resources
     * using this method rather than to rely on a finalization
     * process which may not happen for a long period of time.
     * <p>
     * Graphics objects which are provided as arguments to the paint
     * and update methods of Components are automatically disposed
     * by the system when those methods return.  Programmers should,
     * for efficiency, call the dispose method when finished using
     * a Graphics object only if it was created directly from a
     * Component or another Graphics object.
     *
     * @see       #create(int, int, int, int)
     * @see       #finalize()
     * @see       Component#getGraphics()
     * @see       Component#paint(Graphics)
     * @see       Component#update(Graphics)
     * @since     1.0
     */
    public abstract void dispose();

    /**
```

Beispiel

```
     * Disposes of this graphics context once it is no longer
     * referenced.
     *
     * @see        #dispose()
     * @since      1.0
     */
    public void finalize() {
    dispose();
    }
}
```

Vorlesung 14

Handout S. 6