XMLBeanReader.java

```

import java.beans.*;
import java.io.*;
import org.w3c.dom.*;
import com.ibm.xml.parser.*;
import java.lang.reflect.*;
import java.util.*;
import TestBean1;

public class XMLBeanReader {

  public static void p(String s) {
    System.out.print(s);
  }

  public static void P(String s) {
    System.out.println(s);
  }

  public static void pe(String s) {
    System.err.print(s);
  }

  public static void Pe(String s) {
    System.err.println(s);
  }

  public static void D(String s) {
    Pe(s);
  }

  //
  // XMLBeanReader
  //
  public XMLBeanReader() {
  }

  // Use the Introspector to return an array of property descriptors for
  // bean class c.
  protected static PropertyDescriptor[] getPropertyDescriptors(Class c)
       throws IntrospectionException {

         BeanInfo   beaninfo = Introspector.getBeanInfo(c);
         return beaninfo.getPropertyDescriptors();
  }

  // Setting a property of a bean
  public static void setProperty(Object jb, String sName, Element e,
                          PropertyDescriptor pd) {

    // Merge adjacent text nodes, if any
    e.normalize();

    D("--- SetProperty " + sName);

    // There are plenty of things that can go wrong here, so
    // let's just catch exceptions and report on them.
    try {
```

```
    // We permit property values that can be represented
    // as a String, or property values that are themselves
    // JavaBeans. Here we search all child nodes of this node,
    // looking for nonempty text nodes or a child node
    // of type "JavaBean".
    String value = null;
    Element eJavaBean = null;
    NodeList nl = e.getChildNodes();

    for (int i = 0; i < nl.getLength(); i++) {
      Node n = nl.item(i);

      // Found a text node
      if (n instanceof Text) {
        value = n.getNodeValue();
      }

      // Found a JavaBean node
      if (n instanceof Element) {
        if (n.getNodeName().equals("JavaBean")) {
          eJavaBean = (Element)n;
        }
      }
    }

    D("Value of " + sName + " is '" + value + "'");

    // Get the set accessor for the property. If it's null, there's
    // nothing we can do. We'll use this method somewhere no matter
    // how we set the property.
    D("Getting setter for pd " + pd.toString());
    Method setter = pd.getWriteMethod();
    D("Setter is " + isNull(setter));
    if (setter == null)
      return;

    // Handle indexed and non-indexed property descriptors separately
    if (!(pd instanceof IndexedPropertyDescriptor)) {

      // Is this property of a primitive type? If so, convert the String
      // to that type as appropriate, and call "set" function. Simple.
      Class propType = pd.getPropertyType();
      if (propType == null)
        return;
      D("Property type of " + sName + " is " + propType.toString());

      // This variable will hold the list of arguments to the
      // setter method.
      Object[] setterArgs = null;

      if (propType.isPrimitive()) {
        // All primitive types except for Character have a constructor
        // that uses "String" as an argument.
        if (propType == char.class) {
          char c = (char)(Integer.decode(value).intValue());
          setterArgs = new Object[] { new Character(c) };
        } else {
          // Select wrapper class
          Class wrapper;
          if (propType == boolean.class) wrapper = Boolean.class;
```

```
        else if (propType == byte.class) wrapper = Byte.class;
        else if (propType == int.class) wrapper = Integer.class;
        else if (propType == long.class) wrapper = Long.class;
        else if (propType == float.class) wrapper = Float.class;
        else if (propType == double.class) wrapper = Double.class;
        else {
          D("Can’t find wrapper for " + propType.toString());
          return;
        }

        // Get the constructor for the type that takes a
        // String as an argument
        D("Getting ctor");
        Class[] argTypes = { String.class };
        Constructor ctor = wrapper.getConstructor(argTypes);
        D("ctor is " + isNull(ctor));

        // Create an instance of the object the setter is expecting
        Object[] ctorArgs = { value };
        Object setterArg = ctor.newInstance(ctorArgs);
        D("setterArg is " + isNull(setterArg));

        D("Setting " + sName + " to ’" + setterArg.toString() + "’" +
          " via call to " + setter.toString());

        // Invoke the setter with that argument
        setterArgs = new Object[] { setterArg };
      }
    } else {

      // Handle non-primitive types.
      // 1. If it’s a JavaBean, instantiate the JavaBean it
      // represents, use the new Bean to set the property.
      // 2. If the setter() method for the property takes
      // a single string as its argument, call that setter
      // method.
      // 3. If we can construct a value of the type corresponding
      // to the property’s type, do so, and then call the
      // property’s setter method.
      // Otherwise, we just can’t set this property.
      // So, set the property based on one of these strategies:

      // [1] Is this property’s type represented as a JavaBean
      // in the XML document? The "CLASS" attribute of any
      // JavaBean that appears here must be assignable to the propType
      // in order for this to make sense. Note that the "CLASS"
      // of the bean in the XML tree may be either of the property
      // type OR a subclass of that property type.
      if (eJavaBean != null) {
        String jbClassName = JBClassName(eJavaBean);

        D("jbClassName = " + jbClassName);

        // Check to see if it’s possible to assign the JavaBean
        // class in the document to the class of the property
        // If so, instantiate the JavaBean from the document node
        // and assign the property.
        if (jbClassName != null) {
          D("loading JB");
          Class docBeanClass = Class.forName(jbClassName);

```

```
              if (propType.isAssignableFrom(docBeanClass)) {
                Object argBean = instantiateBean(eJavaBean);

                D("Setting " + sName + " to ’" + argBean.toString() + "’" +
                  " via call to " + setter.toString());

                // Invoke the setter with that argument
                setterArgs = new Object[] { argBean };
              }
            }
          }

          // [2] Does the "setter" function take a single string
          // as its argument? If so, use "value" to set it.
          if (setterArgs == null) {
            Class[] setterParameterTypes = setter.getParameterTypes();
            D("Setter has " + setterParameterTypes.length + " args");

            if (setterParameterTypes.length == 1 &&
                setterParameterTypes[0] == java.lang.String.class) {
                setterArgs = new Object[] { value };
            }
          }

          // [3] Can we create one of these by passing a String to
          // a ctor?
          Constructor ctor = null;
          try {
            ctor = propType.getConstructor(new Class[] { String.class } );
          } catch (Exception exc) {
            // Ignore exceptions
          }

          // If we got a constructor, use it to create the argument
          // to the setter, then call the setter
          if (ctor != null) {
            Object arg = ctor.newInstance(new Object[] { value });
            setterArgs = new Object[] { arg };
          } else {
            // We can’t construct the object with a string
          }
        }

        // At this point we have either figured out what argument
        // to pass to the setter, or we simply don’t know what
        // to do.
        if (setterArgs != null) {
          setter.invoke(jb, setterArgs);
        } else {
          D("Can’t set property " + sName);
        }
      }

      // Handle indexed property types
      else {
      }
    } catch (Exception exc) {
      Pe("setProperty: " + exc.toString());
      Pe("target exception: " + exc.toString());
      exc.printStackTrace();

```

```java
      if (exc instanceof InvocationTargetException) {
        Throwable te = ((InvocationTargetException)exc).getTargetException();
        Pe("-->target exception: " + te.toString());
      }
    }

    D("Finished setProperty(" + sName + ")");
  }

  public static String isNull(Object o) {
    return ((o == null) ? "null" : "not null");
  }

  // Given an XML document element whose type is "JavaBean",
  // return the value of its "CLASS" attribute. Return null
  // if anything goes wrong
  protected static String JBClassName(Element eJavaBean) {
    /// Get bean class name and create bean based on name
    Attr jbClassAttr = eJavaBean.getAttributeNode("CLASS");
    if (jbClassAttr == null)
      return null;
    String jbClassName = jbClassAttr.getValue();
    return jbClassName;
  }

  // Instantiate the JavaBean, and set all of its properties
  // The element must be of type "JavaBean"
  public static Object instantiateBean(Element eJavaBean)
    throws IOException, ClassNotFoundException, IntrospectionException {

    if (!eJavaBean.getNodeName().equals("JavaBean")) {
      D("Can’t get JavaBean name");
      return null;
    }
    D("Trying to get class attr");

    String jbClassName = JBClassName(eJavaBean);
    D("Got jbClassName: " + jbClassName);

    Object jb = Beans.instantiate(null, jbClassName);
    D("Created JavaBean(" + jbClassName + "): " + jb.toString());

    // Get the "Properties" node, which contains all
    // "Property" nodes
    NodeList   kidNodes = eJavaBean.getChildNodes();

    Element eProperties = null;
    for (int i = 0; i < kidNodes.getLength() && eProperties == null; i++) {
      Node n = kidNodes.item(i);
      if (n instanceof Element && n.getNodeName().equals("Properties")) {
        eProperties = (Element)n;
      }
    }

    // Get all "Property" elements for this bean
    NodeList nPropertyNodes = eProperties.getChildNodes();
    D("Got " + nPropertyNodes.getLength() + " child nodes of properties");

    // Create a vector of child Elements which have a tag name "Property"
    Vector vProperties = new Vector();
    for (int i = 0; i < nPropertyNodes.getLength(); i++) {
```

```
      if (nPropertyNodes.item(i) instanceof Element) {
        Element e = (Element)nPropertyNodes.item(i);
        if (e.getNodeName().equals("Property")) {
          vProperties.addElement(e);
        }
      }
    }

    // Use the introspector to get the property descriptors
    // for this bean.
    PropertyDescriptor[] pds = getPropertyDescriptors(jb.getClass());
    D("Got " + pds.length + " property descriptors");

    // Create a hash table of property names and property descriptors
    Hashtable htpd = new Hashtable();
    for (int i = 0; i < pds.length; i++) {
      htpd.put(pds[i].getName().toLowerCase(), pds[i]);
      D("Hashing " + pds[i].getName().toLowerCase() + ": " + isNull(pds[i]));
    }
    D("Done hashing");

    // For each Property element in the nPropertyNodes list,
    // set the appropriate property by calling the
    // appropriate set function. Any properties in the XML for
    // which no PropertyDescriptors exist are ignored.
    // This handles elements like:
    // 3
    for (int i = 0; i < vProperties.size(); i++) {
      try {
        // Property name
        Element e = (Element)(vProperties.elementAt(i));
        String sPropertyName = e.getAttribute("NAME");
        D("Got property " + sPropertyName);

        PropertyDescriptor pd =
          (PropertyDescriptor)htpd.get(sPropertyName.toLowerCase());
        D("PD is " + isNull(pd));
        if (pd == null) {
          D("Couldn't get property descriptor for property " + sPropertyName);
          continue;
        }

        // set the property based on its type, etc.
        setProperty(jb, sPropertyName, e, pd);
      } catch (Exception e) {
        // Any problem that happens here, TOO BAD!
        // This could be changed to throw an exception
        Pe("instantiateBean: " + e.toString());
        e.printStackTrace();
      }
    }

    D("Created new JavaBean of type " + jbClassName);

    // Return the newly-instantiated JavaBean.
    return jb;
  }

  // Read from a file
  public static Object readXMLBean(String s)
    throws IOException, ClassNotFoundException, IntrospectionException {
```

```
    return readXMLBean(new File(s));
  }

  // Read a Bean’s state from an XML file
  public static Object readXMLBean(File f)
    throws IOException, ClassNotFoundException, IntrospectionException {
    FileReader fr = new FileReader(f);
    D("Created file reader");
    Object o = readXMLBean(fr);
    return o;
  }

  // Read a Bean’s state from a character stream
  public static Object readXMLBean(Reader r)
    throws IOException, ClassNotFoundException, IntrospectionException {

    // Read document from XML file
    Parser parser = new Parser("Input");
    D("Created parser");

    Document d = parser.readStream(r);
    D("Got document " + isNull(d));

//    ((TXDocument)d).printWithFormat(new PrintWriter(System.out));

    Element eJavaBean = d.getDocumentElement();
    D("eJavaBean is " + isNull(eJavaBean));

    Object o = instantiateBean(eJavaBean);

    return o;
  }

  // Reads XML from a file, creates the corresponding JavaBean,
  // and then prints the bean by calling its "print()" method (if
  // it has one.)
  public static void main(String args[]) {
    try {
      Object b = readXMLBean(args[0]);
      try {
        Method printer = b.getClass().getMethod("print", null);
        P("--- Read a JavaBean of type " + b.getClass().getName());
        printer.invoke(b, null);
      } catch (Exception ee) {
        P("Unable to print JavaBean");
      }
    } catch (Exception ee) {
      P("Couldn’t read JavaBean from file " + args[0]);
      ee.printStackTrace();
    }
  }
}
```