# Using the JavaSourceFactory

Using the JavaSource framework is using a JavaSourceFactory. This class is (directly or indirectly) the single access point to all other classes. We demonstrate the basic use with the following example:

```
import java.io.File;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.apache.ws.jaxme.js.JavaMethod;
import org.apache.ws.jaxme.js.JavaQName;
import org.apache.ws.jaxme.js.JavaQNameImpl;
import org.apache.ws.jaxme.js.JavaSource;
import org.apache.ws.jaxme.js.JavaSourceFactory;

public class Test {
   public static void main(String[] args) throws Exception {
      // Create a factory
            JavaSourceFactory factory = new JavaSourceFactory();
            // Let the factory create a Java source class "com.mycompany.demo.Parser
              JavaQName className = JavaQNameImpl.getInstance("com.mycompany.demo",
              JavaSource js = factory.newJavaSource(className, "public");
              // Add a method "getParser", which instantiates a new SAXParser
              JavaMethod jm = js.newJavaMethod("getParser", SAXParser.class, "public
              jm.setStatic(true);
              jm.addThrows(ParserConfigurationException.class);
              jm.addLine(SAXParserFactory.class, " sf = ", SAXParserFactory.class, "
              jm.addLine("return sf.newSAXParser();");
              factory.write(null, factory.getLocation(js));
   }
}
```

The example creates a new class `com.mycompany.demo.Parser`. The class has a single, static method `getParser`, which instantiates a new SAX parser. Of couse, you could achieve the same effect much easier with

```
import java.io.File;
import java.io.FileWriter;

public class Test {
   public static void main(String[] args) throws Exception {
      FileWriter fw = new FileWriter("com/mycompany/demo/Parser.java");
```

```
        fw.write("package com.mycompany.demo;\n");
        fw.write("\n");
        fw.write("import javax.xml.parsers.ParserConfigurationException;\n");
        fw.write("import javax.xml.parsers.SAXParser;\n");
        fw.write("import javax.xml.parsers.SAXParserFactory;");
        fw.write("\n");
        fw.write("public class Parser {");
        fw.write("  public SAXParser getParser() throws ParserConfigurationException {
        fw.write("    SAXParserFactory sf = SAXParserFactory.newInstance();");
        fw.write("    return sf.newSAXParser();");
        fw.write("  }");
        fw.write("}");
    }
}
```

The second example has the obvious advantage that it looks better and simpler. However, it is even larger and you've got to care for indentation, imports and the like. You'll never have a chance to postprocess the generated source, unless you are actually parsing it. That's what the JavaSource framework gives you.