# Appendix A – Refactored source code

## Appendix A – Refactored source code

This appendix contains (a) code from the original system, (b) an aspect, and (c) refactored source code, possible due to the aspect.

### 1. Implicit formatting of a new DBHandler object

```
DBHandler serverHandler = new DBHandler();
serverHandler.setFormatter(new DBFormatter());
serverHandler.setLevel(dbLevel);
opLogger.addHandler(serverHandler, SERVER_LOGGER);

DBHandler clientHandler = new DBHandler();
clientHandler.setFormatter(new DBFormatter());
clientHandler.setLevel(dbLevel);
opLogger.addHandler(clientHandler, CLIENT_LOGGER);

DBHandler databaseHandler = new DBHandler();
databaseHandler.setFormatter(new DBFormatter());
databaseHandler.setLevel(dbLevel);
opLogger.addHandler(databaseHandler, DATABASE_LOGGER);
```

Figure 1a: Original code.

```
FormatDBHandler.aj
package com.aspects;

import java.util.logging.Level;
import com.common.Config;
import com.log.DBHandler;
import com.log.DBFormatter;

public aspect FormatDBHandler {

	pointcut DBHandlerCall():
		call(DBHandler.new(..))
		&& within(com..*)
		&& !within(FormatDBHandler);

	DBHandler around(): DBHandlerCall() {
		Level dbLevel = null;
		try{
			Config config = Config.getInstance();
			dbLevel = Level.parse(config.getProperty(Config.DB_LOG));
		}
		catch (Exception opex){
			//This should never happen.
		}

		DBHandler dbHandler = proceed();
		dbHandler.setFormatter(new DBFormatter());
		dbHandler.setLevel(dbLevel);
		return dbHandler;
	}
}
```

Figure 1b: An aspect.

```
DBHandler serverHandler = new DBHandler();
DBHandler clientHandler = new DBHandler();
DBHandler databaseHandler = new DBHandler();

opLogger.addHandler(serverHandler, SERVER_LOGGER);
opLogger.addHandler(clientHandler, CLIENT_LOGGER);
opLogger.addHandler(databaseHandler, DATABASE_LOGGER);
```
Figure 1c: New code.

## 2. Implicit handling of non-existing attributes

```
public Attributes getAttributes() {
	if ( !vec.isEmpty() ){
		return vec.lastElement();
	}
	else{
		return new AttributesImpl();
	}
}


public Attributes getAttributes() {
	return attr;
}
```

Figure 2a: Original code.

```
GetAttributes.aj
package com.aspects;

import org.xml.sax.*;
import org.xml.sax.helpers.*;

public aspect GetAttributes {

	pointcut getAttributes():
		execution(Attributes getAttributes());

	Attributes around(): getAttributes() {
		Attributes result;
		try {
			result = proceed();
		} catch (Exception e) {
			result = new AttributesImpl();
		}
		return result;
	}
}
```

Figure 2b: An aspect.

```
public Attributes getAttributes() {
	return vec.lastElement();
}


public Attributes getAttributes() {
	return attr;
}
```

Figure 2c: New code.

## 3. Aspect-oriented singleton-pattern

```
public static ChstatePoller getInstance() throws OPException{
       try {
              if (chstatePoller == null){
                     chstatePoller = new ChstatePoller();
              }
       } catch(OPException ope){
              ope.addParam(new ExceptionParam(Level.WARNING, CLASS_NAME,
                      "Fel vid instantiering av ChstatePoller."));
              throw ope;
       } catch (Throwable t){
              throw new OPException(t,
                     new ExceptionParam(Level.WARNING, CLASS_NAME,
                     "Fel vid instantiering av ChstatePoller.") );
       }

       return chstatePoller;
}


public static synchronized TraceConsoleHandler getInstance() {
       return new TraceConsoleHandler();
}
```

Figure 3a: Original code.

```
GetInstance.aj
package com.aspects;
import java.util.Hashtable;

public aspect GetInstance {
       private Hashtable instances = new Hashtable();

       pointcut getInstance():
              execution(static * getInstance())
              && within(com..*)
              && !within(com.database..*);

       Object around(): getInstance() {
              Class instance = thisJoinPoint.getSignature().getDeclaringType();
              synchronized(instances) {
                     if(instances.get(instance) == null) {
                            // Proceed only if we haven't already created an instance
                            instances.put(instance, proceed());
                     }
              }
              return (Object) instances.get(instance);
       }
}
```

Figure 3b: An aspect.

```
public static ChstatePoller getInstance() throws OPException{
       try {
              return new ChstatePoller();
       } catch( OPException ope ){
              ope.addParam(new ExceptionParam(Level.WARNING, CLASS_NAME,
                                           "Fel vid instantiering av ChstatePoller."));
              throw ope;
       } catch (Throwable t) {
              throw new OPException(t, new ExceptionParam(Level.WARNING, CLASS_NAME,
                                           "Fel vid instantiering av ChstatePoller."));
       }
}
```

Figure 3c: New code.

## 4. Implicit handling of non existing record elements

```
public synchronized TraceRecord getLastRecord() {
       TraceRecord[] records = _list.toArray(new TraceRecord[]{});
       if ( 0  < records.length ) {
              return records[records.length-1];
       }
       return null;
}

public synchronized TraceRecord getFirstRecord() {
       TraceRecord[] records = _list.toArray(new TraceRecord[]{});
       if ( 0  < records.length ) {
              return records[0];
       }
       return null;
}
```

Figure 4a: Original code.

```
GetRecord.aj
package com.aspects;
import com.log.gui.*;

public aspect GetRecord {

       pointcut getRecordData():
              execution(* get*Record()) &&
              this(TraceRecords);

       after() throwing(ArrayIndexOutOfBoundsException ex): getRecordData() {
              //return null
       }
}
```

Figure 4b: An aspect.

```
public synchronized TraceRecord getLastRecord() {
       TraceRecord[] records = _list.toArray(new TraceRecord[]{});
       return records[records.length-1];
}

public synchronized TraceRecord getFirstRecord() {
       TraceRecord[] records = _list.toArray(new TraceRecord[]{});
       return records[0];
}
```

Figure 4c: New code.

## 5. Implicit creation of directories

```
File file = new java.io.File(logPath + SERVER_FILE);
file.mkdirs();
file = new java.io.File(logPath + CLIENT_FILE);
file.mkdirs();
file = new java.io.File(logPath + DATABASE_FILE);
file.mkdirs();
```

Figure 5a: Original code.

```
NewFile.aj
package com.aspects;
import java.io.File;

public aspect NewFile {

       pointcut newFile(String path):
              call(File.new(String))
              && !within(com.aspects..*)
              && args(path);

       after(String path) returning: newFile(path) {
              File file = new File(path);
              file.mkdirs();
       }
}
```

Figure 5b: An aspect.

```
File file = new java.io.File(logPath + SERVER_FILE);
file = new java.io.File(logPath + CLIENT_FILE);
file = new java.io.File(logPath + DATABASE_FILE);
```

Figure 5c: New code.

## 6. Implicit exception handling when creating a new XMLHandler

```
try {
	xmlHandler = XMLHandler.newHandler();
} catch(ParserConfigurationException pce) {
	throw new OPException(pce, new ExceptionParam(Level.WARNING, CLASS_NAME,
	"Kunde inte skapa XML parser"));
} catch(SAXException saxe) {
	throw new OPException(saxe, new ExceptionParam(Level.WARNING, CLASS_NAME,
	"Kunde inte skapa XML parser"));
}


try {
	xmlHandler = XMLHandler.newHandler();
} catch(Exception e) {
	throw new OPException(e);
}
```

Figure 6a: Original code.

```
NewXMLHandler.aj
package com.aspects;

import com.util.xml.XMLHandler;
import com.common.ExceptionParam;
import com.common.OPException;
import java.util.logging.Level;
import javax.xml.parsers.*;
import org.xml.sax.*;

public aspect NewXMLHandler {

	pointcut newHandler():
		call(XMLHandler XMLHandler.newHandler())
		&& within(com..*)
		&& !within(com.aspects..*);

	declare soft: ParserConfigurationException: newHandler();
	declare soft: SAXException: newHandler();

	after() throwing(ParserConfigurationException pce)
						throws OPException: newHandler() {
		final String CLASS_NAME =
				thisJoinPoint.getStaticPart().getSignature().getName();
		throw new OPException(pce, new ExceptionParam(Level.WARNING, CLASS_NAME,
			"Kunde inte skapa XML parser (ParserConfigurationException)"));
	}
	after() throwing(SAXException saxe) throws OPException: newHandler() {
		final String CLASS_NAME =
				thisJoinPoint.getStaticPart().getSignature().getName();
		throw new OPException(saxe, new ExceptionParam(Level.WARNING,
				CLASS_NAME, "Kunde inte skapa XML parser (SAXException)"));
	}
}
```

Figure 6b: An aspect.

```
xmlHandler = XMLHandler.newHandler();
```

Figure 6c: New code.

## 7. Implicit exception handling for (third party) TextMessage

```
try {
   category = Category.parse(reqTxtMsg.getStringProperty(QueueDefs.CATEGORY));
} catch(Exception e) {
   // Do nothing, at this pont it's just not available
}
try {
   reqDef = reqTxtMsg.getJMSCorrelationID();
} catch(Exception e) {
   // Do nothing, at this pont it's just not available
}
try {
   reqXML = reqTxtMsg.getText();
} catch(Exception e) {
   // Do nothing, at this pont it's just not available
}
try {
   reqFromAddress = reqTxtMsg.getStringProperty(QueueDefs.FROM_ADDRESS);
   respToAddress = new String(reqFromAddress);
} catch(Exception e) {
   // Do nothing, at this pont it's just not available
}
```

Figure 7a: Original code.

```
TextMessageExceptions.aj
package com.aspects;
import javax.jms.JMSException;

public aspect TextMessageExceptions {

       pointcut textMessageExceptions():
              call(String get*(..))
              && within(com.command.CommandMessage);

       declare soft: JMSException: textMessageExceptions();

       String around(): textMessageExceptions() {
              String result = "";
              try {
                     result = proceed();
              } catch(Exception e) {
                     // Do nothing, at this pont it's just not available
              }
              return result;
       }
}
```

Figure 7b: An aspect.

```
category = Category.parse(reqTxtMsg.getStringProperty(QueueDefs.CATEGORY));
reqDef = reqTxtMsg.getJMSCorrelationID();
reqXML = reqTxtMsg.getText();
reqFromAddress = reqTxtMsg.getStringProperty(QueueDefs.FROM_ADDRESS);
if(reqFromAddress != null) respToAddress = new String(reqFromAddress);
```

Figure 7c: New code.

## 8. Implicit null-array handling

```
public static synchronized String toHex(byte[] b, int len) {
       if(b == null) {
              return "null";
       }
       …

public static synchronized String toHex(short[] s) {
       if(s == null) {
              return "null";
       }
       …

public static synchronized String toHex(char[] c) {
       if(c == null) {
              return "null";
       }
       …

public static synchronized String toHex(int[] i) {
       if(i == null) {
              return "null";
       }
       …

public static synchronized String toHex(long[] l) {
       if(l == null) {
              return "null";
       }
       …
}
```

Figure 8a: Original code.

```
ToHex.aj
package com.aspects;
import com.util.Hex;

public aspect ToHex {

       pointcut toHex(Object obj):
              execution(String Hex+.toHex(*[],..))
              && args(obj, ..);

       String around(Object obj): toHex(obj) {
              if(obj == null)
                     return "null";
              else
                     return proceed(obj);
       }
}
```

Figure 8b: An aspect.

```
public static synchronized String toHex(byte[] b, int len) {
       …

public static synchronized String toHex(short[] s) {
       …

public static synchronized String toHex(char[] c) {
       …

public static synchronized String toHex(int[] i) {
       …

public static synchronized String toHex(long[] l) {
       …
```
Figure 8c: New code.