The Makefile is provided with the module generated from getTemplateForDirectory. In the case of Java server implementation, you need to configure the compilation and installation of a Java JAR file:

Makefile for Java

```
JARFILES=<JARName>
<JARName>_DIRS=<DirectoryBaseToIncludeInJAR>
```

The first thing needed to implement a server, is to import the required classes:

Java Imports

```
// Suggested: import alma.<Module>.<Interface>Impl; //But anything, really
package <ChosenPackage>;

// Base component implementation, including container services and component lifecycle infrastructure
import alma.acs.component.ComponentImplBase;

// Skeleton interface for server implementation
import alma.<Module>.<Interface>Operations;

// Error definitions for catching exceptions
import alma.SYSTEMErr.<ExceptionName>Ex;
import alma.<Interface>Err.<ExceptionName>Ex;

// Error definitions for raising exceptions
import alma.SYSTEMErr.wrappers.AcsJ<ExceptionName>Ex;
import alma.<Interface>Err.wrappers.AcsJ<ExceptionName>Ex;
```

Besides this, there's the need to create the class using the mentioned infrastructure provided by the imported class and interface:

Java Server Class Definition

```
// ClassName usually is <Interface>Impl, but can be anything
public class <ClassName> extends ComponentImplBase implements <Interface>Operations {
```

In the case of Java, there's a helper class. Luckily for us, a template for this class is generated from the IDLs and can be found in **$REPO_PATH/ICD/src/alma/<Module>/<Interface>Impl/<Interface>ComponentHelper.java.tpl** after compiling the module. This template has to be copied to your own module and in whatever package you chouse (although is recommended to use the same that was generated). If this is respected, and the package and class naming conventions for your own implementation respected the suggested values, then there's no change needed in the file **(besides removing the .tpl extension)**. In case you decided to use your own conventions look for these parts of the template and edit them accordingly:

This step is necessary only if you didn't copy the .../<Interface>ComponentHelper.java.tpl helper into your package.

Java Server Class Definition

```
// Replace package alma.<Module>.<Interface>Impl; //for:
package alma.<PackageChosenForComponentHelper>;

// Replace import alma.Observatory.<Interface>Impl.<Interface>Impl; //for:
import alma.<PackageChosenForComponentImpl>.<ClassNameForComponentImpl>;

// Replace return new <Interface>Impl(); // for
return new <ClassNameForComponentImpl>();
```

To access a struct, enum, typedef or definition in a module or in an interface, simply do the following:

Java Types Usage

```
// From IDL <Module>::<EnumName>::<VALUE>
import alma.<Module>.<EnumName>;
<EnumName>.<VALUE>;

// From IDL <Module>::<Interface>::<Enumname>::<VALUE>
import alma.<Module>.<Interface>Package.<EnumName>;
<EnumName>.<VALUE>;

// Types
import alma.TYPES.<TypeName>;
```

To retrieve a component to interact with, simply do the following:

Java Component Interaction

```
// Shared
import alma.<Module>.<Interface>;
import alma.<Module>.<Interface>Helper;

// By Name
<Interface> comp = <Interface>Helper.narrow(this.m_containerServices.getComponent("<Name>"));

// By Interface. Must be at least one component configured as default!
<Interface> comp = <Interface>Helper.narrow(this.m_containerServices.getDefaultComponent("IDL:alma/<Module>
/<Interface>:1.0"));

// Release Components
m_containerServies.releaseComponent(comp.name());
```

For logging, as you saw in the class constructor, there's a facility provided by the component. To use it, simply do as follows:

Java Logger

```
m_logger.finer("...");
m_logger.fine("...");
m_logger.info("...");
m_logger.warning("...");
m_logger.severe("...");
```

For catching and raising exceptions:

Java Error Handling

```
// For catching exceptions
import alma.<Interface>Err.<ExceptionName>Ex;
catch (<ExceptionName>Ex e) {

// For raising exceptions
import alma.<Interface>Err.wrappers.AcsJ<ExceptionName>Ex;
throw new AcsJ<ExceptionName>Ex("<CustomMessage>").to<ExceptionName>Ex();

// For raising exceptions with parameters
import alma.<Interface>Err.wrappers.AcsJ<ExceptionName>Ex;
AcsJ<ExceptionName>Ex err = new AcsJ<ExceptionName>Ex("<CustomMessage>");
err.set<ParamName>(<Value>);
throw err.to<ExceptionName>Ex();

// For logging an error message from the exceptions
import alma.<Interface>Err.wrappers.AcsJ<ExceptionName>Ex;
AcsJ<ExceptionName>Ex err = <Interface>ErrImpl.<ExceptionName>ExImpl()
err.log(m_logger);
throw err.to<ExceptionName>Ex();
```

Data types mapping:

To check how data types are mapped between IDL and implementation click here.