# Chapter 1

# Assignment

In this chapter we will present the class that has been assigned to us. First, the lines of code to be analysed are presented without any comment. Afterwards, a brief description of the class role and function is presented. Obviously this is made by us and it is based only on the code and on the documentation provided by the authors.

## 1.1 Assigned class

The class assigned to us is called "**SecurityMechanismSelector**" and is located in the following path relative to the root of Glassfish project:
*appserver/security/ejb.security/src/main/java/com/sun/enterprise/iiop/security/*

The package which the class belongs to is: *com.sun.enterprise.iiop.security*
To present more clearly the class we include the first lines of code containing the documentation of the class and its declaration:

```
108 /**
109  * This class is responsible for making various decisions for selecting
110  * security information to be sent in the IIOP message based on target
111  * configuration and client policies.
112  * Note: This class can be called concurrently by multiple client threads.
113  * However, none of its methods need to be synchronized because the methods
114  * either do not modify state or are idempotent.
115  *
```

```
116  * @author Nithya Subramanian
117
118  */
119
120 @Service
121 @Singleton
122 public final class SecurityMechanismSelector implements PostConstruct {
```

Listing 1.1: Class documentation and declaration

### 1.1.1 Functional role

As the documentation states, this class is responsible for selecting the security mechanism based on target configuration and client policies. It reads the data related to a client and to a target from the current running context of the application.

## 1.2 Assigned methods

The methods assigned to us are 4, that are presented below, with the entire code and with our description of their functional role.

### 1.2.1 First method: getSubjectFromSecurityCurrent

```
963     private Subject getSubjectFromSecurityCurrent()
964             throws SecurityMechanismException {
965         com.sun.enterprise.security.SecurityContext sc = null;
966         sc = com.sun.enterprise.security.SecurityContext.getCurrent();
967         if(sc == null) {
968             if(_logger.isLoggable(Level.FINE)) {
969           _logger.log(Level.FINE," SETTING GUEST ---");
970             }
971             sc = com.sun.enterprise.security.SecurityContext.init();
972         }
973         if(sc == null) {
974             throw new SecurityMechanismException("Could not find " +
975                         " security information");
976         }
977         Subject s = sc.getSubject();
978         if(s == null) {
```

```
979             throw new SecurityMechanismException("Could not find " +
980                 " subject information in the security context.");
981         }
982         if (_logger.isLoggable(Level.FINE)) {
983             _logger.log(Level.FINE, "Subject in security current:" + s);
984         }
985         return s;
986     }
```

Listing 1.2: First assigned method

This method is supposed to return the subject of the current security context. A Subject represents a grouping of related information for a single entity, such as a person. Such information includes the Subject's identities as well as its security-related attributes (passwords and cryptographic keys, for example).

### 1.2.2 Second method: selectSecurityMechanism

```
999     private CompoundSecMech selectSecurityMechanism(
1000                 CompoundSecMech[] mechList) throws SecurityMechanismException {
1001         // We should choose from list of compound security mechanisms
1002         // which are in decreasing preference order. Right now we select
1003         // the first one.
1004         if(mechList == null || mechList.length == 0) {
1005             return null;
1006         }
1007         CompoundSecMech mech = null;
1008         for(int i = 0; i < mechList.length; i++) {
1009             mech = mechList[i];
1010             boolean useMech = useMechanism(mech);
1011             if(useMech) {
1012                 return mech;
1013             }
1014         }
1015         throw new SecurityMechanismException("Cannot use any of the " +
1016             " target's supported mechanisms");
1017     }
```

Listing 1.3: Second assigned method

As the documentation states, this method selects and returns the first[1] supported compound security mechanism from an array given as parameter. The

[1]It returns only the first supported security mechanism found in the array because the **for** cycle stops with **return** instruction when found

mechanism to use is retrieved by calling the method *useMechanism(CompoundSecMech mech)*. If no security mechanism of the array can be used an exception is thrown.

### 1.2.3 Third method: useMechanism

```
1019    private boolean useMechanism(CompoundSecMech mech) {
1020        boolean val = true;
1021        TLS_SEC_TRANS tls = getCtc().getSSLInformation(mech);
1022
1023        if (mech.sas_context_mech.supported_naming_mechanisms.length > 0
1024                && !isMechanismSupported(mech.sas_context_mech)) {
1025            return false;
1026        } else if (mech.as_context_mech.client_authentication_mech.length > 0
1027                && !isMechanismSupportedAS(mech.as_context_mech)) {
1028            return false;
1029        }
1030
1031        if(tls == null) {
1032            return true;
1033        }
1034        int targetRequires = tls.target_requires;
1035        if(isSet(targetRequires, EstablishTrustInClient.value)) {
1036            if(! sslUtils.isKeyAvailable()) {
1037                val = false;
1038            }
1039        }
1040        return val;
1041    }
```

Listing 1.4: Third assigned method

This method checks whether a security mechanism (given as parameter) can be used in the communication process between the client and the target or not. The method returns the boolean value **true** if the client request respects the target configuration, i.e. if the security mechanism required is supported by the target system, there are no errors and if the client can use the protocol TLS (Transport Layer Security) when the target system requires it for the desired security mechanism. Otherwise, if any of the conditions above is not satisfied, the method returns the boolean value **false**.

### 1.2.4 Fourth method: evaluate_client_conformance_ssl

```
1086    private boolean evaluate_client_conformance_ssl(
1087                        EjbIORConfigurationDescriptor iordesc,
1088                        boolean  ssl_used,
1089                        X509Certificate[] certchain)
1090    {
1091      try {
1092        if(_logger.isLoggable(Level.FINE)) {
1093      _logger.log(Level.FINE,
1094      "SecurityMechanismSelector.evaluate_client_conformance_ssl->:");
1095  }
1096
1097        boolean ssl_required  = false;
1098        boolean ssl_supported = false;
1099        int ssl_target_requires = 0;
1100        int ssl_target_supports = 0;
1101
1102        /**************************************************************************
1103         * Conformance Matrix:
1104         *
1105         * |---------------|---------------------|---------------------|------------|
1106         * | SSLClientAuth | targetrequires.ETIC | targetSupports.ETIC | Conformant |
1107         * |---------------|---------------------|---------------------|------------|
1108         * |     Yes       |          0          |      1              |    Yes     |
1109         * |     Yes       |          0          |      0              |    No      |
1110         * |     Yes       |          1          |      X              |    Yes     |
1111         * |     No        |          0          |      X              |    Yes     |
1112         * |     No        |          1          |      X              |    No      |
1113         * |---------------|---------------------|---------------------|------------|
1114         *
1115         **************************************************************************/
1116
1117        // gather the configured SSL security policies.
1118
1119        ssl_target_requires = this.getCtc().getTargetRequires(iordesc);
1120        ssl_target_supports = this.getCtc().getTargetSupports(iordesc);
1121
1122        if (    isSet(ssl_target_requires, Integrity.value)
1123             || isSet(ssl_target_requires, Confidentiality.value)
1124             || isSet(ssl_target_requires, EstablishTrustInClient.value))
1125            ssl_required = true;
1126        else
1127            ssl_required = false;
1128
1129        if ( ssl_target_supports != 0)
1130            ssl_supported = true;
1131        else
1132            ssl_supported = false;
1133
1134        /* Check for conformance for using SSL usage.
```

```
1135            *
1136            * a. if SSL was used, then either the target must require or support
1137            *    SSL. In the latter case, SSL is used because of client policy.
1138            *
1139            * b. if SSL was not used, then the target must not require it either.
1140            *    The target may or may not support SSL (it is irrelevant).
1141            */
1142           if(_logger.isLoggable(Level.FINE)) {
1143         _logger.log(Level.FINE,
1144         "SecurityMechanismSelector.evaluate_client_conformance_ssl:"
1145         + " " + isSet(ssl_target_requires, Integrity.value)
1146         + " " + isSet(ssl_target_requires, Confidentiality.value)
1147         + " " + isSet(ssl_target_requires, EstablishTrustInClient.value)
1148         + " " + ssl_required
1149         + " " + ssl_supported
1150         + " " + ssl_used);
1151     }
1152
1153           if (ssl_used) {
1154               if (! (ssl_required || ssl_supported))
1155                   return false;  // security mechanism did not match
1156           } else {
1157               if (ssl_required)
1158                   return false;  // security mechanism did not match
1159           }
1160
1161           /* Check for conformance for SSL client authentication.
1162            *
1163            * a. if client performed SSL client authentication, then the target
1164            *    must either require or support SSL client authentication. If
1165            *    the target only supports, SSL client authentication is used
1166            *    because of client security policy.
1167            *
1168            * b. if SSL client authentication was not used, then the target must
1169            *    not require SSL client authentcation either. The target may or may
1170            *    not support SSL client authentication (it is irrelevant).
1171            */
1172
1173           if(_logger.isLoggable(Level.FINE)) {
1174         _logger.log(Level.FINE,
1175         "SecurityMechanismSelector.evaluate_client_conformance_ssl:"
1176         + " " + isSet(ssl_target_requires, EstablishTrustInClient.value)
1177         + " " + isSet(ssl_target_supports, EstablishTrustInClient.value));
1178     }
1179
1180           if (certchain != null) {
1181               if ( ! ( isSet(ssl_target_requires, EstablishTrustInClient.value)
1182                        || isSet(ssl_target_supports, EstablishTrustInClient.value)))
1183               return false; // security mechanism did not match
```

```
1184          } else {
1185              if (isSet(ssl_target_requires, EstablishTrustInClient.value))
1186                  return false; // security mechanism did not match
1187          }
1188
1189          if(_logger.isLoggable(Level.FINE)) {
1190        _logger.log(Level.FINE,
1191        "SecurityMechanismSelector.evaluate_client_conformance_ssl: true");
1192    }
1193
1194          return true ; // mechanism matched
1195        } finally {
1196      if(_logger.isLoggable(Level.FINE)) {
1197          _logger.log(Level.FINE,
1198          "SecurityMechanismSelector.evaluate_client_conformance_ssl<-:");
1199      }
1200        }
1201      }
```

Listing 1.5: Fourth assigned method

This method evaluates the conformance of the use of the protocol SSL (Secure Sockets Layer) in the authentication process between the client and the target. The client can ask the authentication via SSL to the target. The target may support or not the SSL authentication and may strictly require it or not.
There are five possible cases in which the client authentication request is conformant to the target configuration or not, based on the possible conditions described before. This conformance cases as described in a table inside a comment block from line 1102 and 1115, in the method's code above. The first column of the table shows whether the client asks for the SSL authentication or not, the second column shows whether the target requires it or not, the third columns shows whether the target supports SSL or not and finally the fourth column shows whether the client request is conformant to the target configuration or not.
The returned boolean value of the method is **true** if the client request for SSL is conformant, **false** otherwise.