UNIVERSITY of STIRLING

CSCU9YW

(Report)

[2410516]

Table of Contents

1. Outline of Problem

1.1 Description and Assumptions

The objective of retrieving music tracks from a database, is achieved using a SOAP (Simple Object Access Protocol) based web service that connects to the database and retrieves music tracks based on composer names and disc numbers. The client created connects to the web service whenever the user pressed the “check” buttons on the client window.

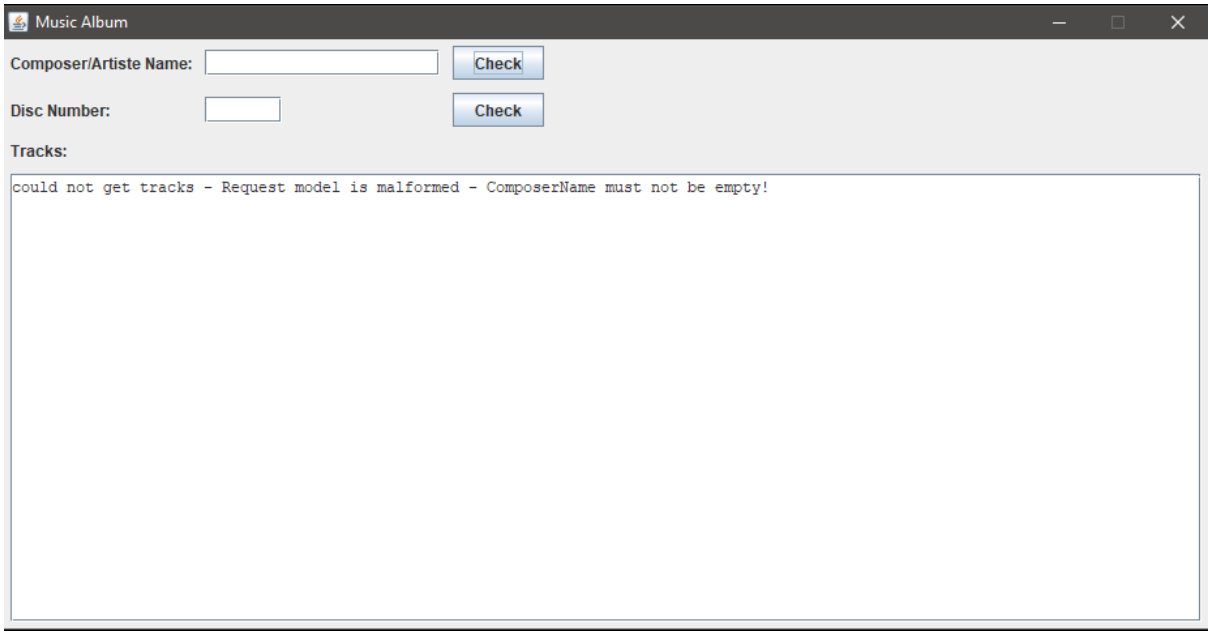
With the creation of the WSDL (Web Services Description Language) file, Axis2’s[1] WSDL2JAVA tool[2] is used to convert the WSDL service file into Java – creating outline client code and outline server code to be used in Java code. The complex types that are created, are mapped to Java classes and then used within the web service as object classes.

No major assumptions are made throughout the development.

1.2 Solution with screenshots of test cases

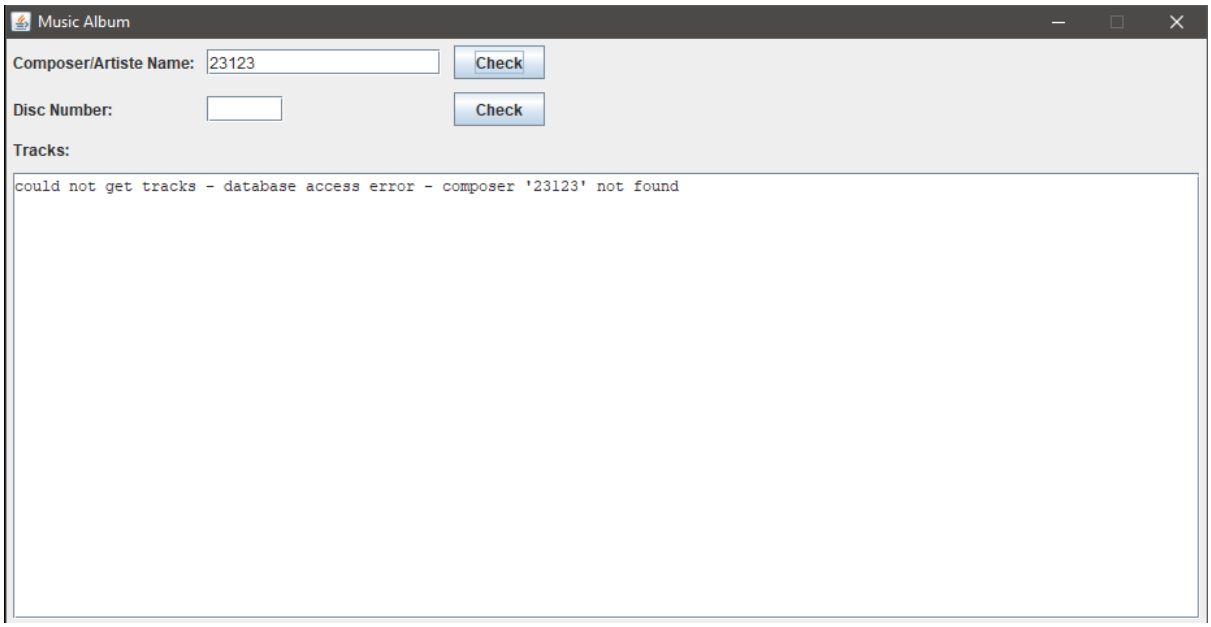
In this sub-section, a series of screenshots will be shown, along with explanations of different test case scenarios. In the end an example of the request and response models will be shown too.

1.2.1 Empty composer name text field



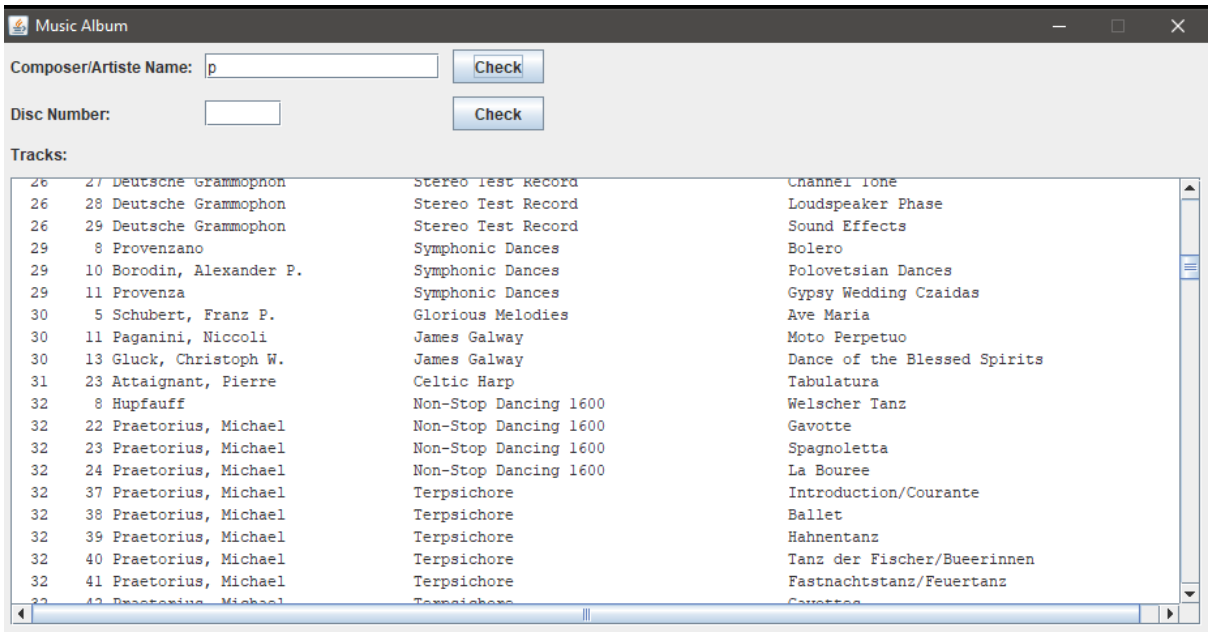
As shown in the image above: When the composer text field is left empty, and the corresponding check button is pressed, an error is outputted by the server to say that the “ComposerName” should not be empty.

1.2.2 Integer entered in composer name text field



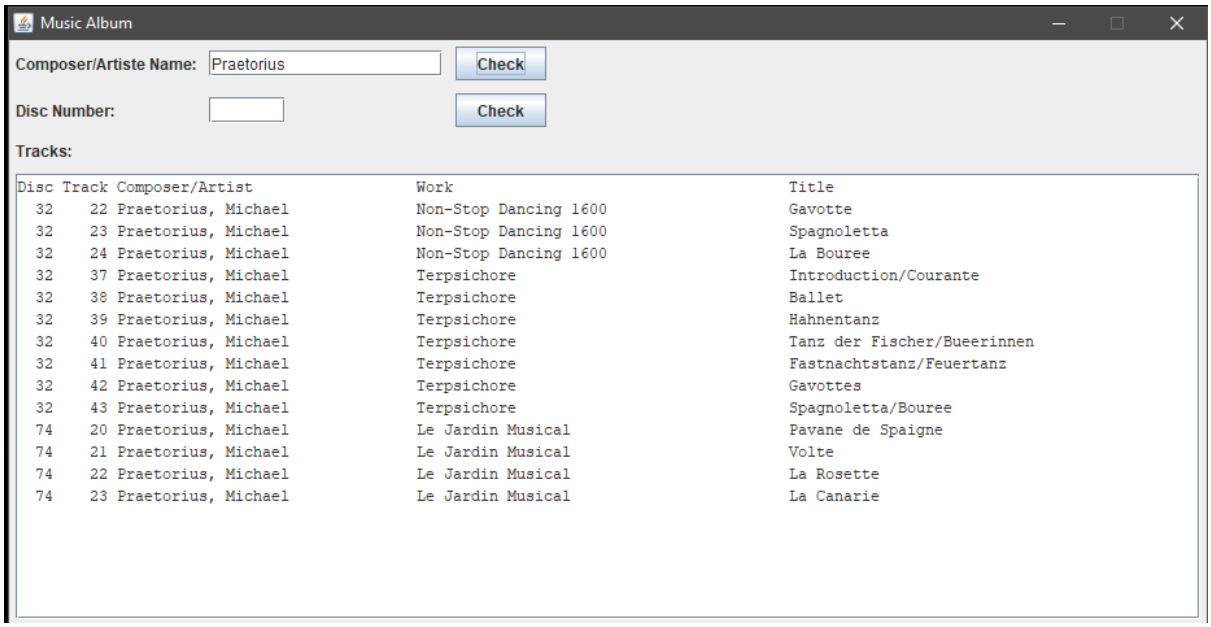
As shown in the image above: When the composer text field is an integer only, the web service still searches for a composer that may have a number in their name as part of their known artist name. In the case above, no composer is found with the specific number and therefore, a not found error is displayed in the client window.

1.2.3 Searching by finding specific characters in composer name



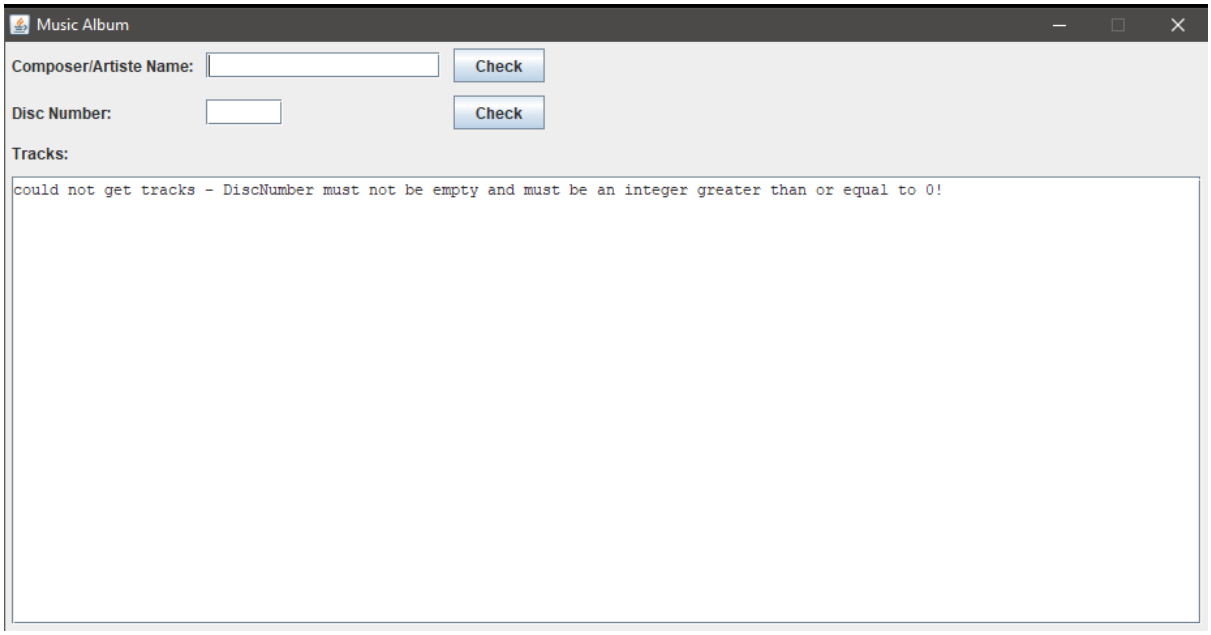
As shown in the image above: The web service searches for composer names that contain the letter “p” in their composer name. In other cases, if a full word was inputted then it would do the same with the word and look for composer names with the word in their name. In this case, the letter “p” returns many results.

1.2.4 Finding track by composer name



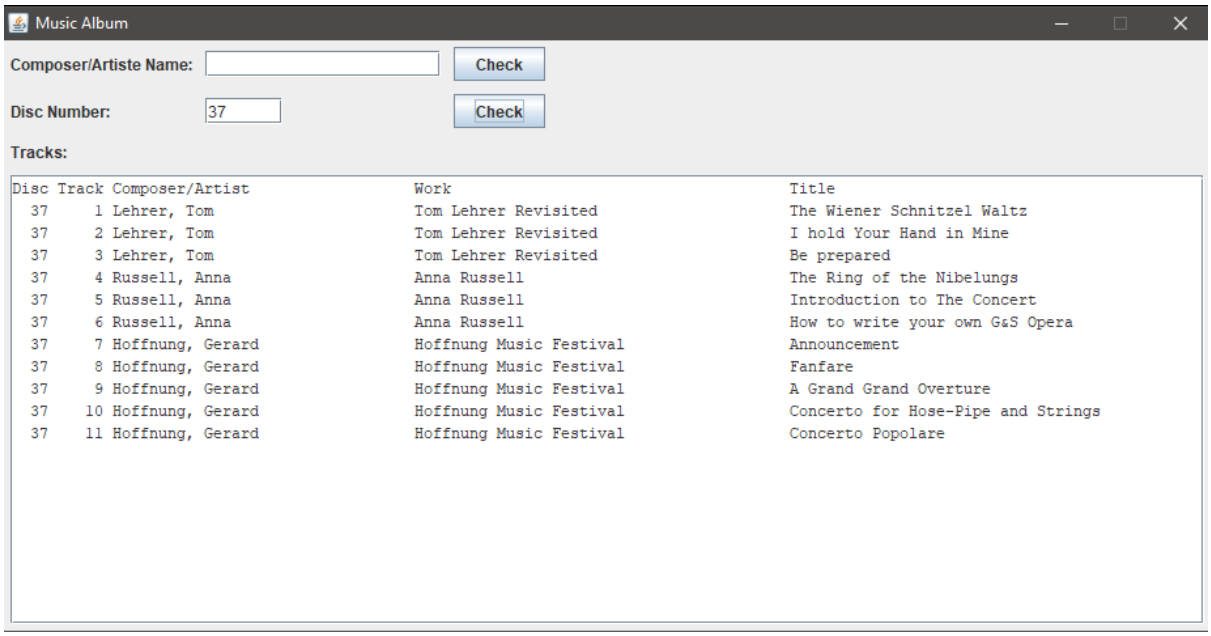
As shown in the image above: Searching for the composer name “Praetorius”, it returns all tracks that contain the string “Praetorius” in their composer name.

1.2.5 Empty disc number field



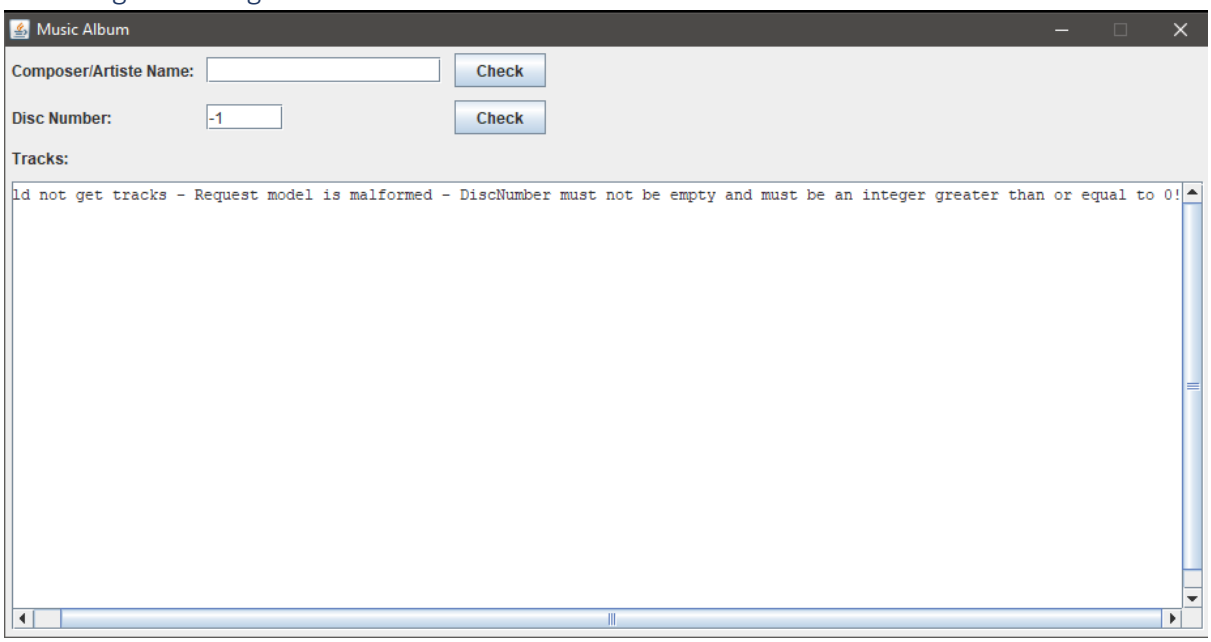
As shown in the image above: When the disc number text field is left blank, an error is displayed to tell the web service user to enter an integer greater than or equal to zero.

1.2.6 Disc number entered in disc text field



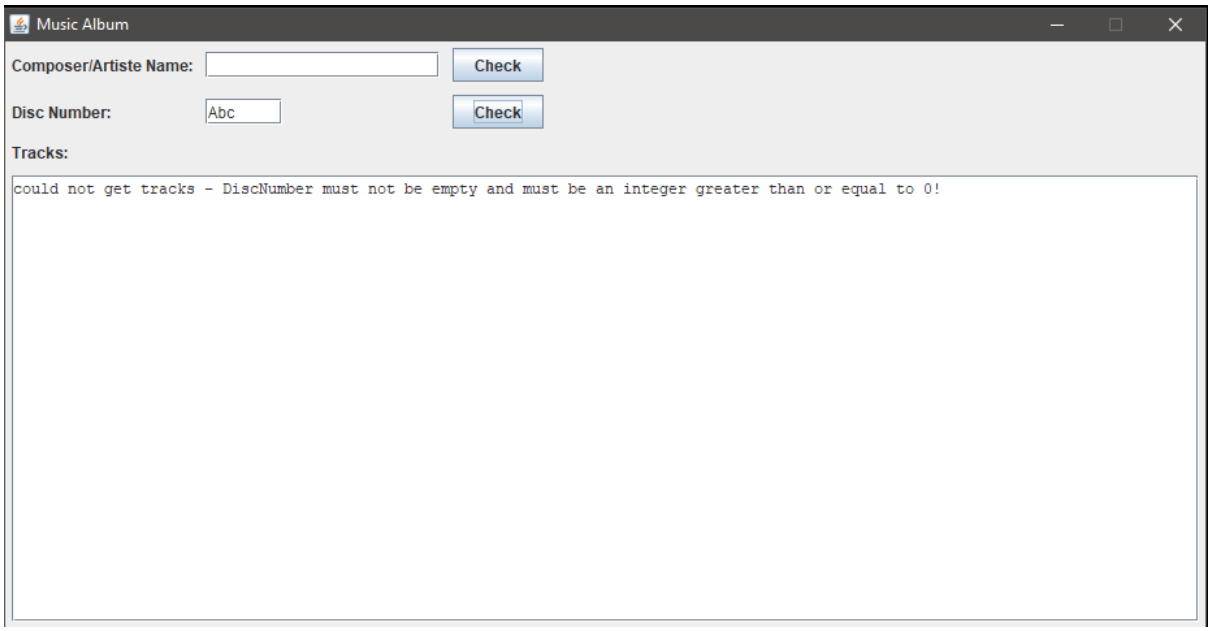
As shown in the image above: The disc number entered is “37” and therefore returns all the tracks within the disc number “37”.

1.2.7 Negative integer in disc number text field



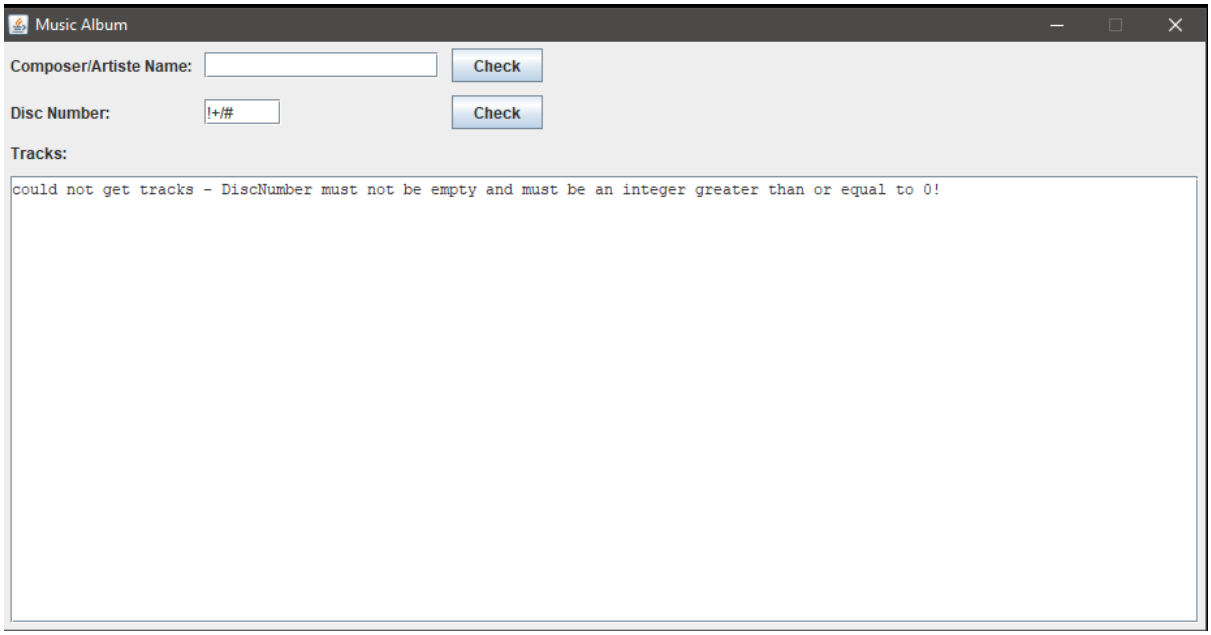
As shown in the image above: The integer being less than zero, returns an error to say that the request model is malformed, with appropriate explanation of what the request model should contain.

1.2.8 String entered in disc number text field



As shown in the image above: The disc number should be an integer greater than or equal to zero. In the case above, a string is inputted and therefore the web service user is warned to use an appropriate integer.

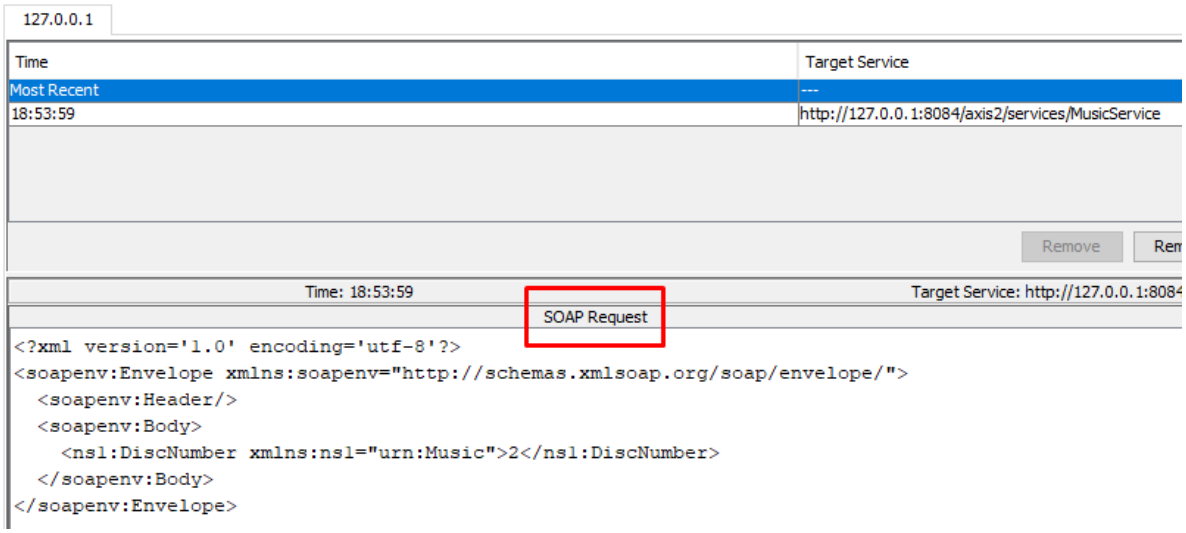
1.2.9 Special character error handling in text fields

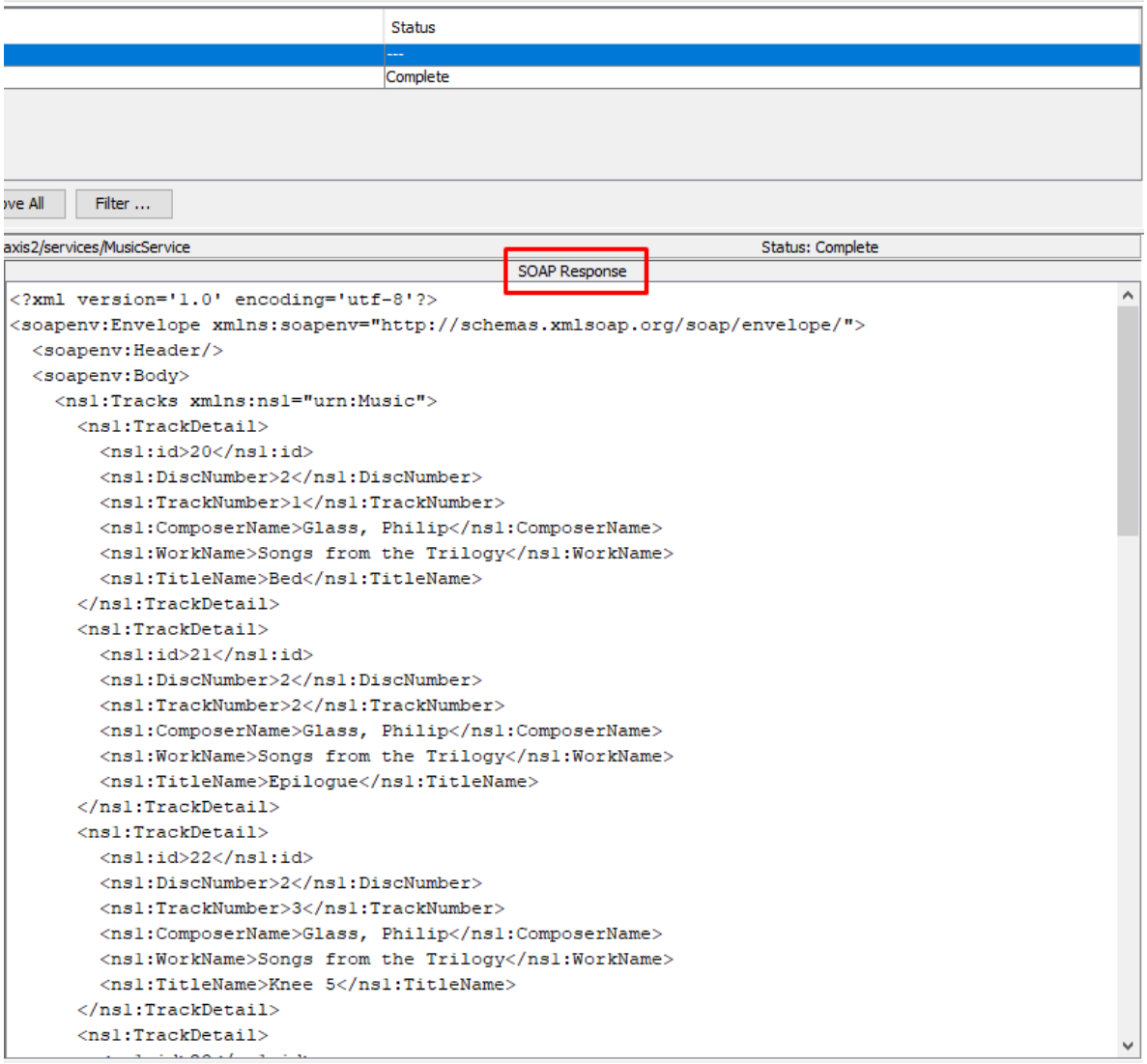


As shown in the image above: Some web services will throw exceptions due to special characters being entered, the text fields and sent along with the requests. Both the composer and disc text fields will handle special characters by asking the web service user to use appropriate values.

1.2.10 SOAP Monitor request and response example

Student Number: 2410516





We can see in the above two images the request and response results shown in the Axis 2 SOAP Monitor [3]. This shows what the user has sent to the web service as a request, and what the web service has responded with, a response with the appropriate complex types.

2. Project completion

2.1 Completed implementation

All requirements were tackled and implemented fully – therefore, all functionality from the assignment sheet have been completed and added to the web service. In addition to the functionality, test cases have been thought of and explained in the screenshots in the previous chapter. The test cases were created before developing the web service and allowed strenuous manual-testing of the project in the end.

Implementation includes:

- SOAP web service using Document/Literal.
- GetByComposerName operation.
- GetByDiscNumber operation.
- Client implementation to allow testing of the web service.
- Documentation of functions and operations within the different parts of code.
- Implementation of SOAP faults, exceptions to be thrown upon issues within the web service and client requests and results. This displays correctly within the client and shows feedback if no music results are found.
- SOAP Monitor testing to ensure correct requests and responses.
- Refactored the code so that the composer name is searched for by looking for composer names that 'contain' a certain value, and that the disc number is searched for with an exact match. This has been changed as the supplied code was searching for disc numbers but returning results from other disc numbers too, rather than exact matches. See MusicService.java code in Appendix 2 for more details, clearly commented too.

2.2 Incomplete implementation and special cases

There are no incomplete implementation and no special cases to be considered.

References

[1] (2018/11/16). *Apache Axis2 – Apache Axis2/Java - Next Generation Web Services*. Available: https://axis.apache.org/axis2/java/core/.

[2] (2018/11/16). *Apache Axis2 – Apache Axis2 User's Guide- Creating Clients*. Available: http://axis.apache.org/axis2/java/core/docs/userguide-creatingclients.html.

[3] (2018/11/16). *Apache Axis2 – The SOAP Monitor Module*. Available: https://axis.apache.org/axis2/java/core/docs/soapmonitor-module.html.

Appendices

Appendix 1: Services.wsdl file

```
<!-- Student Number: 2410516 -->
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MusicDefinitions" targetNamespace="urn:Music"
    xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:music="urn:Music"
    xmlns:wsaw="https://www.w3.org/2006/05/addressing/wsdl"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <types>
        <!-- Schema with elements -->
        <xs:schema targetNamespace="urn:Music"
            xmlns="http://www.w3.org/2001/XMLSchema" xmlns:music="urn:Music"
            attributeFormDefault="qualified" elementFormDefault="qualified">
            <!-- Track Detail -->
            <complexType name="TrackDetail">
                <sequence>
                    <element name="id" type="int" nillable="false" />
                    <element name="DiscNumber" type="int" nillable="true" />
                    <element name="TrackNumber" type="int" nillable="true" />
                    <element name="ComposerName" type="string" nillable="true" />
                    <element name="WorkName" type="string" nillable="true" />
                    <element name="TitleName" type="string" nillable="true" />
                </sequence>
            </complexType>
            <!-- TrackDetails -->
            <complexType name="TrackDetails">
                <sequence>
                    <element name="TrackDetail" type="music:TrackDetail"
                        minOccurs="0" maxOccurs="unbounded" nillable="false" />
                </sequence>
            </complexType>
            <element name="Track" type="music:TrackDetail" nillable="true" />
            <element name="Tracks" type="music:TrackDetails" nillable="true" />
            <element name="ComposerName" type="string" nillable="false" />
            <element name="DiscNumber" type="int" nillable="false" />
            <element name="ErrorMessage" type="string" nillable="true" />
        </xs:schema>
    </types>
    <!-- Messages -->
    <message name="GetByComposerMessage">
        <part name="parameters" element="music:ComposerName" />
    </message>
    <message name="GetByDiscMessage">
        <part name="parameters" element="music:DiscNumber" />
    </message>
    <message name="ResultMessage">
        <part name="return" element="music:Tracks" />
    </message>
    <message name="ErrorFault">
        <part name="return" element="music:ErrorMessage" />
    </message>
    <!-- Operations (with input/output/fault) assigned to musicPort -->
    <portType name="MusicPort">
        <!-- getByComposer operation -->
        <operation name="GetByComposer">
            <documentation>
                This operation takes part of the name for a
                composer/artist, and returns a list of track details whose
```

```
57                composer/artist contains the given string. An "errorFault" is
58                thrown
59                if the name is empty or does not match.
60            </documentation>
61            <input message="music:GetByComposerMessage" wsaw:Action="music:ComposerName" />
62            <output message="music:ResultMessage" wsaw:Action="music:Tracks" />
63            <fault name="ErrorFault" message="music:ErrorFault"
64                wsaw:Action="music:ErrorMessage" />
65        </operation>
66        <!-- getByDisc operation -->
67        <operation name="GetByDisc">
68            <documentation>
69                This operation takes the number of a disc, and returns
70                a list of track details for this disc. An "errorFault" is thrown if
71                the
72                disc number is invalid or does not match.
73            </documentation>
74            <input message="music:GetByDiscMessage" wsaw:Action="music:DiscNumber" />
75            <output message="music:ResultMessage" wsaw:Action="music:Tracks" />
76            <fault name="ErrorFault" message="music:ErrorFault"
77                wsaw:Action="music:ErrorMessage" />
78        </operation>
79    </portType>
80    <!-- Binding of musicBinding using document/literal -->
81    <binding name="MusicBinding" type="music:MusicPort">
82        <soap:binding style="document"
83            transport="http://schemas.xmlsoap.org/soap/http" />
84        <!-- getByComposer operation -->
85        <operation name="GetByComposer">
86            <soap:operation
87                soapAction="http://127.0.0.1:8084/axis2/services/MusicService/GetByComposer"
  />
88            <input>
89                <soap:body use="literal" />
90            </input>
91            <output>
92                <soap:body use="literal" />
93            </output>
94            <fault name="ErrorFault">
95                <soap:fault use="literal" />
96            </fault>
97        </operation>
98        <!-- getByDisc operation -->
99        <operation name="GetByDisc">
100           <soap:operation
101               soapAction="http://127.0.0.1:8084/axis2/services/MusicService/GetByDisc" />
102           <input>
103               <soap:body use="literal" />
104           </input>
105           <output>
106               <soap:body use="literal" />
107           </output>
108           <fault name="ErrorFault">
109               <soap:fault use="literal" />
110           </fault>
111       </operation>
112   </binding>

113   <!-- Music service -->
114   <service name="MusicService">
115       <documentation>The client can remotely enquire about music items by
116           composer/artist name or by disc number. The server responds to
117           queries with reference to the music database.
118       </documentation>
119       <!-- Music service defined with relevant port and binding onto specific
120           address -->
121       <port name="MusicPort" binding="music:MusicBinding">
122           <!-- Address URI: http://127.0.0.1:8083/axis2/services/MusicService -->
123           <soap:address location="http://127.0.0.1:8084/axis2/services/MusicService" />
124       </port>
125   </service>
126</definitions>
```

Appendix 2: MusicService.java file

```
 2 * Student Number: 2410516
 4 package music;
 5
 6 import java.sql.Connection;
10
11 public class MusicService extends MusicServiceSkeleton {
12
13     private final static String databaseHost = "mysql0.cs.stir.ac.uk";
14     private final static String databaseName = "CSCU9YW";
15     private final static String databasePassword = "Password!123";
16     private final static String databaseUser = "ial";
17     private final static String discTable = "music";
18
19     /***
20      * Get tracks by composer name.
21      *
22      * @param trackDetail
23      *            with composerName filled in.
24      * @return tracks relevant to what has been searched for.
25      */
26     public Tracks getByComposer(ComposerName composerName) throws ErrorFault {
27         String fieldName = "composer";
28         String value = composerName.getComposerName();
29
30         if (value != null && !value.isEmpty() && !value.trim().isEmpty()) {
31             try {
32                 TrackDetail[] trackDetails = getByField(fieldName, value, false);
33                 TrackDetails trackDetailsList = new TrackDetails();
34                 trackDetailsList.setTrackDetail(trackDetails);
35                 Tracks tracks = new Tracks();
36
37                 tracks.setTracks(trackDetailsList);
38
39                 return tracks;
40             } catch (Exception ex) {
41                 throw ex;
42             }
43         } else {
44             Exception exception = new Exception("ComposerName must not be empty!");
45             String errorMessage = "Request model is malformed - " + exception.getMessage();
46             throw (new ErrorFault(errorMessage, exception));
47         }
48     }
49
50     /***
51      * Get tracks by disc number.
52      *
53      * @param trackDetail
54      *            with composerName filled in.
55      * @return tracks relevant to what has been searched for.
56      */
57     public Tracks getByDisc(DiscNumber discNumber) throws ErrorFault {
58         String fieldName = "disc";
59         int value = discNumber.getDiscNumber();
60
61         if (value >= 0) {
62             try {
```

```
63                TrackDetail[] trackDetails = getByField(fieldName, Integer.toString(value),
  true);
64                TrackDetails trackDetailsList = new TrackDetails();
65                trackDetailsList.setTrackDetail(trackDetails);
66                Tracks tracks = new Tracks();
67
68                tracks.setTracks(trackDetailsList);
69
70                return tracks;
71            } catch (NumberFormatException ex) {
72                throw (new ErrorFault("DiscNumber must not be empty and must be an integer
  greater than or equal to 0!"
73                        ex));
74            } catch (Exception ex) {
75                throw ex;
76            }
77        } else {
78            Exception exception = new Exception(
79                    "DiscNumber must not be empty and must be an integer greater than or equal
  to 0!");
80            String errorMessage = "Request model is malformed - " + exception.getMessage();
81            throw (new ErrorFault(errorMessage, exception));
82        }
83    }
84
85    /**
86     * Get value from database, based on field name being the column name and
87     * the value.
88     *
89     * @param field
90     *            name to be equal to the column name from the database
91     * @param value
92     *            of the field value from the database
93     * @param isExactMatch
94     *            should be set to false if the value in the database should
95     *            contain the value parameter
96     * @return array of TrackDetail objects
97     * @throws ErrorFault
98     *             if exception occurs
99     */
100   private TrackDetail[] getByField(String field, String value, boolean isExactMatch) throws
  ErrorFault {
101       try {
102           // Check field and value are not empty
103           if (field.length() == 0)
104               throw (new Exception("field is empty"));
105           if (value.length() == 0)
106               throw (new Exception("value is empty"));
107           // Form a database connection
108           Class.forName("com.mysql.jdbc.Driver").newInstance();
109           String databaseDesignation = "jdbc:mysql://" + databaseHost + "/" + databaseName +
  "?user=" + databaseUser
110                   + "&password=" + databasePassword;
111           Connection connection = DriverManager.getConnection(databaseDesignation);
112           Statement statement = connection.createStatement();
113           // Query to find non-exact match
114           String query = "SELECT id, disc, track, composer, work, title " + "FROM " +
```

```
    discTable + " " + "WHERE "
115                     + field + " LIKE '%" + value + "%'";
116             // Query to find exact match
117             if (isExactMatch) {
118                 query = "SELECT id, disc, track, composer, work, title " + "FROM " + discTable
    + " " + "WHERE " + field
119                         + " = '" + value + "'";
120             }
121             ResultSet result = statement.executeQuery(query);
122             result.last();
123             int resultCount = result.getRow();
124             // Throw exception if result is not found
125             if (resultCount == 0)
126                 throw (new Exception(field + " '" + value + "' not found"));
127
128             TrackDetail[] trackDetails = new TrackDetail[resultCount];
129             result.beforeFirst();
130             int resultIndex = 0;
131             while (result.next()) {
132                 TrackDetail trackDetail = new TrackDetail();
133
134                 // Id
135                 trackDetail.setId(result.getInt(1));
136                 // Disc number
137                 trackDetail.setDiscNumber(result.getInt(2));
138                 // Track number
139                 trackDetail.setTrackNumber(result.getInt(3));
140                 // Composer name
141                 trackDetail.setComposerName(result.getString(4));
142                 // Work name
143                 trackDetail.setWorkName(result.getString(5));
144                 // Title name
145                 trackDetail.setTitleName(result.getString(6));
146
147                 // Add the TrackDetail to the list of TrackDetail objects
148                 trackDetails[resultIndex++] = trackDetail;
149             }
150
151             connection.close();
152
153             return (trackDetails);
154         } catch (Exception exception) {
155             String errorMessage = "database access error - " + exception.getMessage();
156             throw (new ErrorFault(errorMessage, exception));
157         }
158     }
159 }
160
```

Appendix 3: Client.java file

```
 2 * Student Number: 2410516
 4 package music;
 5
 6 import music.MusicServiceStub;
12
13 public class Client extends JFrame implements ActionListener {
14     // Final variables
15     private final static int contentInset = 5;
16     private final static int trackColumns = 130;
17     private final static int trackRows = 20;
18     private final static int gridLeft = GridBagConstraints.WEST;
19     private final static String programTitle = "Music Album";
20
21     // UI variables
22     private GridBagConstraints contentConstraints = new GridBagConstraints();
23     private GridBagLayout contentLayout = new GridBagLayout();
24     private Container contentPane = getContentPane();
25     private JButton discButton = new JButton("Check");
26     private JLabel discLabel = new JLabel("Disc Number:");
27     private JTextField discText = new JTextField(5);
28     private JButton nameButton = new JButton("Check");
29     private JLabel nameLabel = new JLabel("Composer/Artiste Name:");
30     private JTextField nameText = new JTextField(16);
31     private Font trackFont = new Font(Font.MONOSPACED, Font.PLAIN, 12);
32     private JLabel trackLabel = new JLabel("Tracks:");
33     private JTextArea trackArea = new JTextArea(trackRows, trackColumns);
34     private JScrollPane trackScroller = new JScrollPane(trackArea);
35
36     // Tracks returned
37     private Tracks tracks;
38     // MusicServiceStub
39     private MusicServiceStub musicServiceStub;
40
41     public Client() throws Exception {
42         // Setup UI
43         contentPane.setLayout(contentLayout);
44         addComponent(0, 0, gridLeft, nameLabel);
45         addComponent(1, 0, gridLeft, nameText);
46         addComponent(2, 0, gridLeft, nameButton);
47         addComponent(0, 1, gridLeft, discLabel);
48         addComponent(1, 1, gridLeft, discText);
49         addComponent(2, 1, gridLeft, discButton);
50         addComponent(0, 2, gridLeft, trackLabel);
51         addComponent(0, 3, gridLeft, trackScroller);
52         // Add action listeners
53         nameButton.addActionListener(this);
54         discButton.addActionListener(this);
55         trackArea.setFont(trackFont);
56         trackArea.setEditable(false);
57         // Initialise stub
58         musicServiceStub = new MusicServiceStub();
59     }
60
61     public static void main(String[] args) throws Exception {
62         // Get screen dimensions
63         Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
64         int screenWidth = screenSize.width;
```

```
65          int screenHeight = screenSize.height;
66
67          // Initialise client
68          Client window = new Client();
69          // Window setup
70          window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
71          window.setTitle(programTitle);
72          window.setResizable(false);
73          window.pack();
74          int windowWidth = window.getWidth();
75          int windowHeight = window.getHeight();
76          int windowX = (screenWidth - windowWidth) / 2;
77          int windowY = (screenHeight - windowHeight) / 2;
78          window.setLocation(windowX, windowY);
79          window.setVisible(true);
80      }
81
82      /**
83       * Add component to contentPane of window
84       *
85       * @param x
86       *            coordinate to position component
87       * @param y
88       *            coordinate to position component
89       * @param position
90       *            of component
91       * @param component
92       *            that needs adding
93       */
94      private void addComponent(int x, int y, int position, JComponent component) {
95          Insets contentInsets = new Insets(contentInset, contentInset, contentInset,
    contentInset);
96          contentConstraints.gridx = x;
97          contentConstraints.gridy = y;
98          contentConstraints.anchor = position;
99          contentConstraints.insets = contentInsets;
100         if (component == trackArea || component == trackLabel)
101             contentConstraints.gridwidth = GridBagConstraints.REMAINDER;
102         contentLayout.setConstraints(component, contentConstraints);
103         contentPane.add(component);
104     }
105
106     /**
107      * Action performed.
108      *
109      * @param event
110      *            that has occurred.
111      */
112     public void actionPerformed(ActionEvent event) {
113         String trackRows = "";
114         TrackDetail[] tracks;
115         try {
116             if (event.getSource() == nameButton)
117                 tracks = getField("composer", nameText.getText());
118             else if (event.getSource() == discButton)
119                 tracks = getField("disc", discText.getText());
120             else
```

```
121                 return
122             // Add to display on screen the table row column names
123             trackRows += String.format("%4s %5s %-32s %-40s %-40s\n", "Disc", "Track",
    "Composer/Artist", "Work",
124                     "Title");
125             for (int i = 0; i < tracks.length; i++) {
126                 TrackDetail trackDetail = tracks[i];
127
128                 // Add to display on screen in the table
129                 trackRows += String.format("%4s %5s %-32s %-40s %-40s\n",
    trackDetail.getDiscNumber(),
130                         trackDetail.getTrackNumber(), trackDetail.getComposerName(),
    trackDetail.getWorkName(),
131                         trackDetail.getTitleName());
132             }
133         } catch (ErrorFault exception) {
134             String error = exception.getMessage();
135             if (error == null)
136                 error = exception.toString();
137             error = "could not get tracks - " + error;
138             trackRows += error;
139         }
140
141         trackArea.setText(trackRows);
142     }
143
144     /**
145      * Get field value based on field name and the value.
146      *
147      * @param field
148      *            name from database
149      * @param value
150      *            that the field value should be equal to
151      * @return Array of TrackDetail objects
152      * @throws ErrorFault
153      *             when an exception occurs
154      */
155     private TrackDetail[] getField(String field, String value) throws ErrorFault {
156         // Switch on the field name
157         // Can easily add new fields in switch case block
158         switch (field) {
159
160         // Composer field
161         case "composer":
162             ComposerName composerName = new ComposerName();
163
164             try {
165                 composerName.setComposerName(value);
166                 tracks = musicServiceStub.getByComposer(composerName);
167             } catch (ErrorFault exception) {
168                 throw exception;
169             } catch (RemoteException exception) {
170                 throw (new ErrorFault(exception.getMessage(), exception));
171             }
172
173             return tracks.getTracks().getTrackDetail();
174
```

```
175         // Disc field
176         case "disc"
177             DiscNumber discNumber = new DiscNumber()
178
179             try {
180                 discNumber.setDiscNumber(Integer.parseInt(value))
181                 tracks = musicServiceStub.getByDisc(discNumber)
182             } catch (ErrorFault exception) {
183                 throw exception
184             } catch (NumberFormatException ex) {
185                 throw (new ErrorFault("DiscNumber must not be empty and must be an integer
    greater than or equal to 0!"
186                         ex))
187             } catch (RemoteException exception) {
188                 throw (new ErrorFault(exception.getMessage(), exception))
189             }
190
191             return tracks.getTracks().getTrackDetail()
192         default
193             // Returns null if the field name is not in the switch case
194             // statement
195             return null
196
197         }
198     }
199
200 }
201
```