## Table of Contents

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

# Chapter 5. Working with Complex Data Types

In the previous chapter, we created RPC-style services in Java. Those services dealt only with simple data types. Simple data types may suffice in many situations, but you'll eventually need to use more complex data types, like those you're familiar with in your day-to-day Java programming. In this chapter we'll look at creating services with method parameters and return values that are arrays and Java beans. We'll hold off on custom data types until the next chapter, since it's possible that any custom types you create would use the complex data types we'll be discussing here.

## 5.1. Passing Arrays as Parameters

Let's face it — arrays are probably the most common complex data type in programming. They're everywhere, so their use in SOAP is critical. We covered the details of SOAP arrays back in Chapter 3, so you should be aware of how arrays are encoded. So let's get right into writing some Java code for services that use arrays.

We've been working with stock market examples, so let's stick with that theme. It might be useful to have a service that returns information about a collection of stock symbols. It might provide the total volume of shares traded for the day, the average trading price of those stocks, the number of stocks trading higher for the day, etc. There are lots of possibilities. Let's start out with a service that returns the total number of shares traded for the day. The service is called `urn:BasicTradingService`, and it has a method called `getTotalVolume`. Here is the Java class that implements the service:

## 5.2. Returning Arrays

So far we've been passing arrays as parameters. Now let's use an array as the return value of a service method. We'll add a method to our service called `getMostActive( )`, which returns a `String[]` that contains the symbols for the most actively traded stocks of the day. Here's the new version of the `BasicTradingService` class:

```
package javasoap.book.ch5;
public class BasicTradingService {

   public BasicTradingService(  ) {
   }
   public String[] getMostActive(  ) {
      // get the most actively traded stocks
      String[] actives = { "ABC", "DEF", "GHI", "JKL" };
      return actives;
   }
   public int getTotalVolume(String[] stocks) {

      // get the volumes for each stock from some
      // data feed and return the total
      int total = 345000;
      return total;
   }
   public String executeTrade(Object[] params) {
      String result;
      try {
         String stock = (String)params[0];
         Integer numShares = (Integer)params[1];
         Boolean buy = (Boolean)params[2];
         String orderType = "Buy";
         if (false == buy.booleanValue(  )) {
            orderType = "Sell";
         }
         result = (orderType + " " + numShares + " of " + stock);
      }
      catch (ClassCastException e) {
         result = "Bad Parameter Type Encountered";
      }
      return result;
   }
}
```

## 5.3. Passing Custom Types as Parameters

As Java programmers, we certainly don't restrict ourselves to the classes found in the standard Java packages. A substantial part of our code exists as custom types, Java classes of our own creation that embody the functionality and characteristics of the systems we're building.

Consider the design of a Java class that contains all of the data necessary to specify a stock trade. This new class might contain the symbol for the stock being traded, the number of shares to trade, and an indication of the order type (buy or sell). When designing such a class, it's important to view it in the context of the larger system. That kind of analysis yields clues that can lead to decisions regarding the properties and behaviors to be given to the class. We all do this kind of work all the time; it's called software design. The result is a Java class that contains methods for accessing properties and behavior. Since SOAP is a data transport, we're interested in the properties of the class. That's what we want to transmit over the wire.

## 5.4. Returning Custom Types

It's equally useful (and equally common) to return custom types from service method calls. We can enhance our trading service by offering a method that takes a single stock symbol as a parameter and returns its high and low trading prices for the day. The classes `HighLow_ServerSide` and `HighLow_ClientSide` represent the high/low prices on the server and client, respectively.

```
package javasoap.book.ch5;
public class HighLow_ServerSide {
   public float _high;
   public float _low;
   public HighLow_ServerSide(  ) {
   }
   public HighLow_ServerSide (float high, float low) {
      setHigh(high);
      setLow(low);
   }
   public float getHigh(  ) {
      return _high;
   }
   public void setHigh(float high) {
      _high = high;
   }
   public float getLow(  ) {
      return _low;
   }
   public void setLow(float low) {
      _low = low;
   }
}

package javasoap.book.ch5;
public class HighLow_ClientSide {
   public float _high;
   public float _low;
   public String toString(  ) {
      return "High: " + _high +
        " Low: " + _low;
   }
   public HighLow_ClientSide(  ) {
   }
   public float getHigh(  ) {
      return _high;
   }
   public void setHigh(float high) {
      _high = high;
   }
   public float getLow(  ) {
      return _low;
   }
   public void setLow(float low) {
      _low = low;
   }
}
```