

Exceptions in CORBA IDL: CORBA IDL allows exceptions to be defined in interfaces and thrown by their methods. To illustrate this point, we have defined our list of shapes in the server as a sequence of a fixed length (line 4) and have defined *FullException* (line 6), which is thrown by the method *newShape* (line 7) if the client attempts to add a shape when the sequence is full.

Invocation semantics: Remote invocation in CORBA has *at-most-once* call semantics as the default. However, IDL may specify that the invocation of a particular method has *maybe* semantics by using the *oneway* keyword. The client does not block on *oneway* requests, which can be used only for methods without results. For an example of a *oneway* request, see the example on callbacks at the end of Section 17.2.1.

**The CORBA Naming service**  $\diamond$  The CORBA Naming Service is discussed in Section 17.3.1. It is a binder that provides operations including *rebind* for servers to register the remote object references of CORBA objects by name and *resolve* for clients to look them up by name. The names are structured in a hierarchic fashion, and each name in a path is inside a structure called a *NameComponent*. This makes access in a simple example seem rather complex.

**CORBA pseudo objects (Java 2 version 1.4)**  $\diamond$  Implementations of CORBA provide some interfaces to the functionality of the ORB that programmers need to use. They are called pseudo-objects because they cannot be used like CORBA objects; for example, they cannot be passed as arguments in RMIs. They have IDL interfaces and are implemented as libraries. Those relevant to our simple example are:

- The ORB interface includes: The method *init*, which must be called to initialize the ORB; the method *resolve\_initial\_references*, which is used to find services such as the Naming Service and the root POA; other methods, which enable conversions between remote object references and strings.
- The POA (Portable Object Adaptor – see Figure 17.6 and page 678) interface includes: A method for activating a POA manager; a method *servant\_to\_reference* for registering a CORBA object.

### 17.2.1 CORBA client and server example (for Java 2 version 1.4)

This section outlines the steps necessary to produce client and server programs that use the IDL *Shape* and *ShapeList* interfaces shown in Figure 17.1. This is followed by a discussion of callbacks in CORBA. We use Java as the client and server languages, but the approach is similar for other languages. The interface compiler *idlj* can be applied to the CORBA interfaces to generate the following items:

- The equivalent Java interfaces – two per IDL interface. For example, the interfaces *ShapeListOperations* and *ShapeList* are shown in Figure 17.2.
- The server skeletons for each *idl* interface. The names of skeleton classes end in *POA*, for example *ShapeListPOA*.
- The proxy classes or client stubs, one for each IDL interface. The names of these classes end in *Stub*, for example *\_ShapeListStub*.

**Figure 17.2** Java interfaces generated by *idlj* from CORBA interface *ShapeList*.

```

public interface ShapeListOperations {
    Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException;
    Shape[] allShapes();
    int getVersion();
}

public interface ShapeList extends ShapeListOperations, org.omg.CORBA.Object,
org.omg.CORBA.portable.IDLEntity { } // interface ShapeList

```

- A Java class to correspond to each of the *structs* defined with the IDL interfaces. In our example, classes *Rectangle* and *GraphicalObject* are generated. Each of these classes contains a declaration of one instance variable for each field in the corresponding *struct* and a pair of constructors, but no other methods.
- Classes called helpers and holders, one for each of the types defined in the IDL interface. A helper class contains the *narrow* method, which is used to cast down from a given object reference to the class to which it belongs, which is lower down the class hierarchy. For example, the *narrow* method in *ShapeHelper* casts down to class *Shape*. The holder classes deal with *out* and *inout* arguments, which cannot be mapped directly onto Java. See Exercise 17.9 for an example of the use of holders.

**Server program** ◊ The server program should contain implementations of one or more IDL interfaces. For a server written in an object-oriented language such as Java or C++, these implementations are implemented as servant classes. CORBA objects are instances of servant classes.

When a server creates an instance of a servant class, it must register it with the POA, which makes the instance into a CORBA object and gives it a remote object reference. Unless this is done, it will not be able to receive remote invocations. Readers who studied Chapter 5 carefully may realize that registering the object with the POA causes it to be recorded in the CORBA equivalent of the remote object table.

In our example, the server contains implementations of the interfaces *Shape* and *ShapeList* in the form of two servant classes, together with a server class that contains a *initialization* section (see Section 5.2.5) in its *main* method.

*The servant classes:* Each servant class extends the corresponding skeleton class and implements the methods of an IDL interface using the method signatures defined in the equivalent Java interface. The servant class that implements the *ShapeList* interface is named *ShapeListServant*, although any other name could have been chosen. Its outline is shown in Figure 17.3. Consider the method *newShape* in line 1, which is a factory method because it creates *Shape* objects. To make a *Shape* object a CORBA object, it is registered with the POA by means of its *servant\_to\_reference* method, as shown in line 2. Complete versions of the IDL interface and the client and server classes in this example are available at [cdk3.net/corba](http://cdk3.net/corba).

*The server:* The *main* method in the server class *ShapeListServer* is shown in Figure 17.4. It first creates and initializes the ORB (line 1). It gets a reference to the root

**Figure 17.3** *ShapeListServant* class of the Java server program for CORBA interface *ShapeList*

```

import org.omg.CORBA.*;
import org.omg.PortableServer.POA;
class ShapeListServant extends ShapeListPOA {
    private POA theRootpoa;
    private Shape theList[];
    private int version;
    private static int n=0;
    public ShapeListServant(POA rootpoa){
        theRootpoa = rootpoa;
        // initialize the other instance variables
    }
    public Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException {1
        version++;
        Shape s = null;
        ShapeServant shapeRef = new ShapeServant( g, version);
        try {
            org.omg.CORBA.Object ref = theRootpoa.servant_to_reference(shapeRef); 2
            s = ShapeHelper.narrow(ref);
        } catch (Exception e) {}
        if(n >=100) throw new ShapeListPackage.FullException();
        theList[n++] = s;
        return s;
    }
    public Shape[] allShapes(){ ... }
    public int getVersion() { ... }
}

```

POA and activates the POAManager (lines 2 & 3). Then it creates an instance of *ShapeListServant*, which is just a Java object (line 4). It then makes it into a CORBA object by registering it with the POA (line 5). After this, it registers the server with the Naming Service. It then waits for incoming client requests (line 10).

Servers using the Naming Service first get a root naming context (line 6), then make a *NameComponent* (line 7), define a path (line 8) and finally use the *rebind* method (line 9) to register the name and remote object reference. Clients carry out the same steps but use the *resolve* method as shown in Figure 17.5 line 2.

**The client program** ◊ An example client program is shown in Figure 17.5. It creates and initializes an ORB (line 1), then contacts the Naming Service to get a reference to the remote *ShapeList* object by using its *resolve* method (line 2). After that it invokes its method *allShapes* (line 3) to obtain a sequence of remote object references to all the *Shapes* currently held at the server. It then invokes the *getAllState* method (line 4), giving as argument the first remote object reference in the sequence returned; the result is supplied as an instance of the *GraphicalObject* class.

Figure 17.4 Java class *ShapeListServer*

```

import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
public class ShapeListServer {
    public static void main(String args[]) {
        try{
            ORB orb = ORB.init(args, null);           1
            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));2
            rootpoa.the_POAManager().activate();      3
            ShapeListServant shapeRef = new ShapeListServant(rootpoa); 4
            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(SLSRef); 5
            ShapeList SLRef = ShapeListHelper.narrow(ref);
            org.omg.CORBA.Object objRef = orb.resolve_initial_references("Name.Service");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent("ShapeList", ""); 6
            NameComponent path[] = {nc};             7
            ncRef.rebind(path, SLRef);               8
            orb.run();                               9
        } catch (Exception e) { ... }
    }
}

```

The *getAllState* method seems to contradict our earlier statement that objects cannot be passed by value in CORBA, because both client and server deal in instances of the class *GraphicalObject*. However, there is no contradiction: the CORBA object returns a *struct*, and clients using a different language might see it differently. For example, in the C++ language the client would see it as a *struct*. Even in Java, the generated class *GraphicalObject* is more like a *struct* because it has no methods.

Client programs should always catch CORBA *SystemExceptions*, which report on errors due to distribution (see line 5). Client programs should also catch the exceptions defined in the IDL interface, such as the *FullException* thrown by the *newShape* method.

This example illustrates the use of the *narrow* operation: the *resolve* operation of the Naming Service returns a value of type *Object*; this type is narrowed to suit the particular type required – *ShapeList*.

**Callbacks** ◊ Callbacks can be implemented in CORBA in a manner similar to the one described for Java RMI in Section 5.5.1. For example, the *WhiteboardCallback* interface may be defined as follows:

```

interface WhiteboardCallback {
    oneway void callback(in int version);
};

```

Figure 17.5 Java client program for CORBA interfaces *Shape* and *ShapeList*

```

import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class ShapeListClient{
    public static void main(String args[]) {
        try{
            ORB orb = ORB.init(args, null);
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent("ShapeList", "");
            NameComponent path [] = { nc };
            ShapeList shapeListRef =
                ShapeListHelper.narrow(ncRef.resolve(path));
            Shape[] sList = shapeListRef.allShapes();
            GraphicalObject g = sList[0].getAllState();
        } catch(org.omg.CORBA.SystemException e) {...}
    }
}

```

This interface is implemented as a CORBA object by the client, enabling the server to send the client a version number whenever new objects are added. But before the server can do this, the client needs to inform the server of the remote object reference of its object. To make this possible, the *ShapeList* interface requires additional methods such as *register* and *deregister*, as follows:

```

int register(in WhiteboardCallback callback);
void deregister(in int callbackId);

```

After a client has obtained a reference to the *ShapeList* object and created an instance of *WhiteboardCallback*, it uses the *register* method of *ShapeList* to inform the server that it is interested in receiving callbacks. The *ShapeList* object in the server is responsible for keeping a list of interested clients and notifying all of them each time its version number increases when a new object is added. The *callback* method is declared as *oneway* so that the server may use asynchronous calls to avoid delay as it notifies each client.

### 17.2.2 The architecture of CORBA

The architecture is designed to support the role of an object request broker that enables clients to invoke methods in remote objects, where both clients and servers can be implemented in a variety of programming languages. The main components of the CORBA architecture are illustrated in Figure 17.6.