**Figure 17.1** IDL interfaces \textit{Shape} and \textit{ShapeList}.

```idl
struct Rectangle{
    long width;
    long height;
    long x;
    long y;
};

struct GraphicalObject {
    string type;
    Rectangle enclosing;
    boolean isFilled;
};

interface Shape {
    long getVersion();
    GraphicalObject getAllState(); // returns state of the GraphicalObject
};

typedef sequence <Shape, 100> All;

interface ShapeList {
    exception FullException{ }
    Shape newShape(in GraphicalObject g) raises (FullException);
    All allShapes(); // returns sequence of remote object references
    long getVersion();
};
```
Figure 17.2  Java interface `ShapeList` generated by `idltojava` from CORBA interface `ShapeList`.

```java
public interface ShapeList extends org.omg.CORBA.Object {
    Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException;
    Shape[] allShapes();
    int getVersion();
}
```
**Figure 17.3**  
*ShapeListServant* class of the Java server program for CORBA interface *ShapeList*

```java
import org.omg.CORBA.*;
class ShapeListServant extends _ShapeListImplBase {
    ORB theOrb;
    private Shape theList[];
    private int version;
    private static int n=0;
    public ShapeListServant(ORB orb){
        theOrb = orb;
        // initialize the other instance variables
    }
    public Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException {
        version++;
        Shape s = new ShapeServant( g, version);
        if(n >=100) throw new ShapeListPackage.FullException();
        theList[n++] = s;
        theOrb.connect(s);
        return s;
    }
    public Shape[] allShapes(){ ... }
    public int getVersion() { ... }
}
```
Figure 17.4  Java class ShapeListServer

```java
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
public class ShapeListServer {
    public static void main(String args[]) {
        try{
            ORB orb = ORB.init(args, null);
            ShapeListServant shapeRef = new ShapeListServant(orb);
            orb.connect(shapeRef);
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent("ShapeList", ".");
            NameComponent path[] = {nc};
            ncRef.rebind(path, shapeRef);
            java.lang.Object sync = new java.lang.Object();
            synchronized (sync) { sync.wait(); }
        } catch (Exception e) { ... }
    }
}
```
Figure 17.5  Java client program for CORBA interfaces Shape and ShapeList

```java
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
public class ShapeListClient{
    public static void main(String args[])
    {
        try{
            ORB orb = ORB.init(args, null);  // 1
            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));  // 2
            Shape[] sList = shapeListRef.allShapes();  // 3
            GraphicalObject g = sList[0].getAllState();  // 4
            } catch(org.omg.CORBA.SystemException e) {...}
```
Figure 17.6  The main components of the CORBA architecture

client

implementation repository

interface repository

server

ORB core

object adapter

skeleton

Servant A

client program

proxy for A

ORB core

or dynamic invocation

Request

Reply

or dynamic skeleton

core core for A

Servant A
Figure 17.7 IDL module *Whiteboard*.

```
module Whiteboard {
    struct Rectangle {
        ...
    };
    struct GraphicalObject {
        ...
    };
    interface Shape {
        ...
    };
    typedef sequence <Shape, 100> All;
    interface ShapeList {
        ...
    };
};
```
Figure 17.8  IDL constructed types - slide 1.

<table>
<thead>
<tr>
<th>Type</th>
<th>Examples</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence</td>
<td>typedef sequence &lt;Shape, 100&gt; All; typedef sequence &lt;Shape&gt; All</td>
<td>Defines a type for a variable-length sequence of elements of a specified IDL type. An upper bound on the length may be specified.</td>
</tr>
<tr>
<td></td>
<td>bounded and unbounded sequences of Shapes</td>
<td></td>
</tr>
<tr>
<td>string</td>
<td>string name; typedef string&lt;8&gt; SmallString; unbounded and bounded sequences of characters</td>
<td>Defines a sequences of characters, terminated by the null character. An upper bound on the length may be specified.</td>
</tr>
<tr>
<td>array</td>
<td>typedef octet uniqueId[12]; typedef GraphicalObject GO[10][8]</td>
<td>Defines a type for a multi-dimensional fixed-length sequence of elements of a specified IDL type.</td>
</tr>
</tbody>
</table>

*this figure continues on the next slide*
### IDL constructed types - slide 2

<table>
<thead>
<tr>
<th>Type</th>
<th>Examples</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>record</td>
<td>struct GraphicalObject {</td>
<td>Defines a type for a record containing a group of related entities. <em>Structs</em> are passed by value in arguments and results.</td>
</tr>
<tr>
<td></td>
<td>string type;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rectangle enclosing;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>boolean isFilled;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>};</td>
<td></td>
</tr>
<tr>
<td>enumerated</td>
<td>enum Rand</td>
<td>The enumerated type in IDL maps a type name onto a small set of integer values.</td>
</tr>
<tr>
<td></td>
<td>(Exp, Number, Name);</td>
<td></td>
</tr>
<tr>
<td>union</td>
<td>union Exp switch (Rand) {</td>
<td>The IDL discriminated union allows one of a given set of types to be passed as an argument. The header is parameterized by an <em>enum</em>, which specifies which member is in use.</td>
</tr>
<tr>
<td></td>
<td>case Exp: string vote;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>case Number: long n;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>case Name: string s;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>};</td>
<td></td>
</tr>
</tbody>
</table>
**CORBA interoperable object references**

**IOR format**

<table>
<thead>
<tr>
<th>IDL interface type name</th>
<th>Protocol and address details</th>
<th>Object key</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface repository identifier</td>
<td>IIOP</td>
<td>host domain name</td>
</tr>
</tbody>
</table>
Figure 17.9    Naming graph in CORBA Naming Service

initial naming context

```
ShapeList
  C -> D -> E
```

initial naming context

```
P -> Q
  R
```

initial naming context

```
V
  XX
  S -> U -> T
```
Instructor's Guide for Coulouris, Dollimore and Kindberg
Distributed Systems: Concepts and Design Edn. 3    ©Pearson Education 2001

Figure 17.10  Part of the CORBA Naming Service NamingContext interface in IDL

```
struct NameComponent { string id; string kind; };

typedef sequence <NameComponent> Name;

interface NamingContext  {

    void bind (in Name n,  in Object  obj);
    binds the given name and remote object reference in my context.

    void unbind (in Name n);
    removes an existing binding with the given name.

    void bind_new_context(in Name n);
    creates a new naming context and binds it to a given name in my context.

    Object  resolve (in Name n);
    looks up the name in my context and returns its remote object reference.

    void list  (in unsigned long how_many, out BindingList bl, out BindingIterator bi);
    returns the names in the bindings in my context.

};
```
Figure 17.11  CORBA event channels