OMG IDL to Java Language Mapping *1*

Note – The OMG IDL to Java Language Mapping specification is aligned with CORBA version 2.3.2

Note – This document reflects the changes made by the IDL/Java 2k RTF in blue (see Section , “Abstract Interfaces,” on page 1-28 and Section 1.21.5.3, “Portable ObjectImpl,” on page 1-116), and the changes made by the Java/IDL 2k RTF in magenta (see Section , “read_Object,” on page 1-107 and Section , “read_abstract_interface,” on page 1-107). The reference to location of the Java sources in Section 1.1.1, “org.omg.* Packages,” on page 1-3 needs to be updated when this document is formally approved.
Editor: Jeff Mischkinsky, jeff@persistence.com, jeff_mischkinsky@omg.org

This chapter describes the complete mapping of IDL into the Java language. All changebars represent changes made since CORBA 2.3 and 2.3.1. The OMG documents used to update this chapter were ptc/00-01-05 and ptc/00-01-07.

Examples of the mapping are provided. It should be noted that the examples are code fragments that try to illustrate only the language construct being described. Normally they will be embedded in some module and hence will be mapped into a Java package.

Contents

This chapter contains the following sections.

Section Title	Page
"Introduction"	1-2
"Names"	1-4
"Mapping of Module"	1-5
"Mapping for Basic Types"	1-6
"Helpers"	1-13
"Mapping for Constant"	1-16
"Mapping for Enum"	1-17
"Mapping for Struct"	1-20
"Mapping for Union"	1-21
"Mapping for Sequence"	1-24
"Mapping for Array"	1-26
"Mapping for Interface"	1-27
"Mapping for Exception"	1-52
"Mapping for the Any Type"	1-57
"Mapping for Certain Nested Types"	1-61
"Mapping for Typedef"	1-62
"Mapping Pseudo Objects to Java"	1-63
"Server-Side Mapping"	1-88
"Java ORB Portability Interfaces"	1-100

1.1 Introduction

This section describes the complete mapping of IDL into the Java language. It is based upon versions of the Java JDK 1.1 and above.

Examples of the mapping are provided. It should be noted that the examples are code fragments that try to illustrate only the language construct being described. Normally they will be embedded in some module and hence will be mapped into a Java package.

In a variety of places, methods are shown as throwing particular system exceptions. These instances are identified because it was felt that the particular system exception was more likely to occur and hence is called out as a "hint" to callers (and

implementers). Please remember that, except where specifically mandated by this specification, it is implementation dependent (and legal) for the ORB runtime to raise other system exceptions.

There are two cases in which methods in the **org.omg.*** packages are specified as throwing the exception **org.omg.CORBA.NO_IMPLEMENT**:

1. Deprecated methods - Implementations are not mandatory for these methods. They are clearly marked with a javadoc comment.
2. Non abstract methods (not deprecated) - that have been added since the Java language mapping was first approved. To maintain binary compatibility, these methods are not defined as abstract, but as concrete and throw **org.omg.CORBA.NO_IMPLEMENT**. Conforming implementations shall override these methods to provide the specified semantics.

In various places the notation **{...}** is used in describing Java code. This indicates that concrete Java code will be generated for the method body and that the method is concrete, not abstract. Normally the generated code is specific to a particular vendor's implementation and is "internal" to their implementation.

1.1.1 org.omg.* Packages

The Java language mapping is highly dependent upon the specification of a set of standard Java packages contained in **org.omg.***. The complete and exact contents (the Java sources) of all portions of the Java mapping which cannot be mapped using the rules for mapping IDL to Java are specified in an associated zip file, which is available on the OMG's public server as document ptc/00-02-01. This includes Java classes for all PIDL, native types, and ORB portability interfaces. The zip file is the definitive statement of the exact contents of the packages. While every attempt has been made to assure that the text contained in this document is accurate, there may be some minor discrepancies. The zip file is the authoritative statement of the specification because it contains the actual Java code.

It is probable that OMG specifications that are adopted in the future may make changes to these packages (e.g., to add a method to the ORB). Care must be taken to ensure that such future changes do not break binary compatibility with previous versions.

1.1.1.1 Allowable Modifications

Conforming implementations may not add or subtract anything from the definitions contained in the org.omg.* packages except as follows:

- The value for the field **DEFAULT_ORB** in org/omg/CORBA/ORB.java may be changed to indicate the vendor's default ORB implementation.
- The value for the field **sDEFAULT_ORB_SINGLETON** in org/omg/CORBA/ORB.java may be changed to indicate the vendor's default ORB singleton implementation.

- The addition of *javadoc* comments for documenting ORB APIs. Removal of specified *javadoc* comments, in particular comments marking code as deprecated, is forbidden.
- The names of formal parameters for methods for which the entire implementation is provided by the vendor.

1.2 Names

In general IDL names and identifiers are mapped to Java names and identifiers with no change. If a name collision could be generated in the mapped Java code, the name collision is resolved by prepending an underscore (_) to the mapped name.

In addition, because of the nature of the Java language, a single IDL construct may be mapped to several (differently named) Java constructs. The “additional” names are constructed by appending a descriptive suffix. For example, the IDL interface **foo** is mapped to the Java interfaces **foo** and **fooOperations**, and additional Java classes **fooHelper**, **fooHolder**, **fooPOA**, and optionally **fooPOATie**. If more than one reserved suffix is present in an IDL name, than an additional underscore is prepended to the mapped name for each additional suffix.

In those exceptional cases that the “additional” names could conflict with other mapped IDL names, the resolution rule described above is applied to the other mapped IDL names. The naming and use of required “additional” names takes precedence.

For example, an interface whose name is **fooHelper** or **fooHolder** is mapped to **_fooHelper** or **_fooHolder** respectively, regardless of whether an interface named **foo** exists. The helper and holder classes for interface **fooHelper** are named **_fooHelperHelper** and **_fooHelperHolder**.

IDL names that would normally be mapped unchanged to Java identifiers that conflict with Java reserved words will have the collision rule applied.

1.2.1 Reserved Names

The mapping in effect reserves the use of several names for its own purposes. These are:

- The Java class ***<type>*****Helper**, where ***<type>*** is the name of an IDL defined type.
- The Java class ***<type>*****Holder**, where ***<type>*** is the name of an IDL defined type (with certain exceptions such as typedef aliases).
- The Java classes ***<basicJavaType>*****Holder**, where ***<basicJavaType>*** is one of the Java primitive datatypes that is used by one of the IDL basic datatypes (Section 1.4.1.4, “Holder Classes,” on page 1-7).
- The Java classes ***<interface>*****Operations**, ***<interface>POA***, and ***<interface>*****POATie**, where ***<interface>*** is the name of an IDL interface type.
- The nested scope Java package name ***<interface>*****Package**, where ***<interface>*** is the name of an IDL interface (Section 1.17, “Mapping for Certain Nested Types,” on page 1-61).

- The keywords in the Java language: (from the Java Language Specification 1.0 First Edition, Section 3.9)

```
abstract    default   if          private      throw
boolean     do        implements  protected    throws
break       double    import      public       transient
byte        else      instanceof  return       try
case        extends   int         short        void
catch       final     interface   static       volatile
char        finally   long        super        while
class       float     native      switch
const       for       new         synchronized
continue    goto      package     this
```

- The additional Java constants:

```
true        false     null
```

- IDL declarations that collide with the following methods on java.lang.Object (from the Java Language Specification 1.0 First Edition, Section 20.1):

```
clone     equals      finalize    getClass     hashCode
notify    notifyAll   toString    wait
```

The use of any of these names for a user defined IDL type or interface (assuming it is also a legal IDL name) will result in the mapped name having an underscore (_) prepended.

1.3 Mapping of Module

An IDL module is mapped to a Java package with the same name. All IDL type declarations within the module are mapped to corresponding Java class or interface declarations within the generated package.

IDL declarations not enclosed in any modules are mapped into the (unnamed) Java global scope.

1.3.1 Example

```
// IDL
module Example {...}

// generated Java
package Example;
   ...
```

1.4 Mapping for Basic Types

1.4.1 Introduction

Table 1-1 on page 1-6 shows the basic mapping. In some cases where there is a potential mismatch between an IDL type and its mapped Java type, the Exceptions column lists the standard CORBA exceptions that may be (or are) raised. See Section 1.15, “Mapping for Exception,” on page 1-52 for details on how IDL system exceptions are mapped.

The potential mismatch can occur when the range of the Java type is “larger” than IDL. The value must be effectively checked at runtime when it is marshaled as an in parameter (or on input for an inout). For example, Java chars are a superset of IDL chars.

Users should be careful when using unsigned types in Java. Because there is no support in the Java language for unsigned types, a user is responsible for ensuring that large unsigned IDL type values are handled correctly as negative integers in Java.

Table 1-1 Basic Type Mappings

IDL Type	Java type	Exceptions
boolean	`boolean`	
char	`char`	CORBA::DATA_CONVERSION
wchar	`char`	CORBA::DATA_CONVERSION
octet	`byte`	
string	`java.lang.String`	CORBA::MARSHAL CORBA::DATA_CONVERSION
wstring	`java.lang.String`	CORBA::MARSHAL CORBA::DATA_CONVERSION
short	`short`	
unsigned short	`short`	
long	`int`	
unsigned long	`int`	
long long	`long`	
unsigned long long	`long`	
float	`float`	
double	`double`	
fixed	`java.math.BigDecimal`	CORBA::DATA_CONVERSION

1.4.1.1 Future Support

In the future it is expected that the IDL type **long double** will be supported directly by Java. Currently there is no support for this type in JDK 1.1, and as a practical matter, it is not yet widely supported by ORB vendors.

IDL Type	Java type	Exceptions
long double	`not available at this time`	

1.4.1.2 IDLEntity

Many of the Java interfaces and classes generated from IDL are marked by implementing or extending an empty marker interface **`IDLEntity`** which has no methods. The following sections identify the specifics. The IIOP serialization classes specified in the reverse Java to IDL mapping (see the *Java to IDL Language Mapping* document) will detect these instances and marshal them using the generated marshaling code in the Helper class.

```
// Java

package org.omg.CORBA.portable;

interface IDLEntity extends java.io.Serializable {}
```

1.4.1.3 Java Serialization

Those generated classes that are not abstract, including the stub classes, shall support Java object serialization semantics. For example, generated helper classes do not have to be serializable. The following classes support Java object serialization semantics:

- stub classes
- abstract base classes for concrete valuetypes
- implementation classes for concrete valuetypes

1.4.1.4 Holder Classes

Support for out and inout parameter passing modes requires the use of additional "holder" classes. These classes are available for all of the basic IDL datatypes in the **`org.omg.CORBA`** package and are generated for all named user defined IDL types except those defined by typedefs. (Note that in this context user defined includes types that are defined in OMG specifications such as those for the Interface Repository, and other OMG services.)

For user defined IDL types, the holder class name is constructed by appending **`Holder`** to the mapped (Java) name of the type.

For the basic IDL datatypes, the holder class name is the Java type name (with its initial letter capitalized) to which the datatype is mapped with an appended **Holder**, (e.g., **IntHolder**.)

Each holder class has a constructor from an instance, a default constructor, and has a public instance member, **value**, which is the typed value. The default constructor sets the value field to the default value for the type as defined by the Java language: **false** for boolean, **0** for numeric and char types, **null** for strings, null for object references.

To support portable stubs and skeletons, holder classes also implement the **org.omg.CORBA.portable.Streamable** interface.

The holder classes for the basic types are listed below. The complete definition is provided for the first one, ShortHolder. The rest follow the same pattern, simply substituting the proper typename for short. The elided definitions are specified below. Note that they implement the **Streamable** interface as well and are in the **org.omg.CORBA** package.

In addition, holders are generated for some of the CORBA Core PIDL. Types for which this is true are specifically identified in this language mapping specification.

Also note that the holder class for Principal is officially deprecated. See Section 1.19.12, “Principal,” on page 1-87 for more information.

```
// Java

package org.omg.CORBA;

final public class ShortHolder
      implements org.omg.CORBA.portable.Streamable {
   public short value;
   public ShortHolder() {}
   public ShortHolder(short initial) {
      value = initial;
   }
   public void _read(
               org.omg.CORBA.portable.InputStream is) {
      value = is.read_short();
   }
   public void _write(
               org.omg.CORBA.portable.OutputStream os) {
      os.write_short(value);
   }
   public org.omg.CORBA.TypeCode _type() {
      return org.omg.CORBA.ORB.init().get_primitive_tc(
                                    TCKind.tk_short);
   }
}

final public class IntHolder
      implements org.omg.CORBA.portable.Streamable {
   public int value;
```

```
   public IntHolder() {}
   public IntHolder(int initial) {...}
   public void _read(
      org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
      org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class LongHolder
      implements org.omg.CORBA.portable.Streamable{
   public long value;
   public LongHolder() {}
   public LongHolder(long initial) {...}
   public void _read(
      org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
      org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}
final public class ByteHolder
      implements org.omg.CORBA.portable.Streamable{
   public byte value;
   public ByteHolder() {}
   public ByteHolder(byte initial) {...}
   public void _read(
      org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
      org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class FloatHolder
      implements org.omg.CORBA.portable.Streamable{
   public float value;
   public FloatHolder() {}
   public FloatHolder(float initial) {...}
   public void _read(
      org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
      org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class DoubleHolder
      implements org.omg.CORBA.portable.Streamable{
   public double value;
   public DoubleHolder() {}
   public DoubleHolder(double initial) {...}
   public void _read(
      org.omg.CORBA.portable.InputStream is) {...}
```

```
   public void _write(
      org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class CharHolder
      implements org.omg.CORBA.portable.Streamable{
   public char value;
   public CharHolder() {}
   public CharHolder(char initial) {...}
   public void _read(
         org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
         org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class BooleanHolder
      implements org.omg.CORBA.portable.Streamable{
   public boolean value;
   public BooleanHolder() {}
   public BooleanHolder(boolean initial) {...}
   public void _read(
         org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
         org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class StringHolder
      implements org.omg.CORBA.portable.Streamable{
   public java.lang.String value;
   public StringHolder() {}
   public StringHolder(java.lang.String initial) {...}
   public void _read(
         org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
         org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class ObjectHolder
      implements org.omg.CORBA.portable.Streamable{
   public org.omg.CORBA.Object value;
   public ObjectHolder() {}
   public ObjectHolder(org.omg.CORBA.Object initial) {...}
   public void _read(
         org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
         org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
```

```
}

final public class AnyHolder
   implements org.omg.CORBA.portable.Streamable{
   public Any value;
   public AnyHolder() {}
   public AnyHolder(Any initial) {...}
   public void _read(
         org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
         org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class TypeCodeHolder
   implements org.omg.CORBA.portable.Streamable{
   public TypeCode value;
   public typeCodeHolder() {}
   public TypeCodeHolder(TypeCode initial) {...}
   public void _read(
         org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
         org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

/**
*@deprecated Deprecated by CORBA 2.2.
*/
final public class PrincipalHolder
   implements org.omg.CORBA.portable.Streamable{
   public Principal value;
   public PrincipalHolder() {}
   public PrincipalHolder(Principal initial) {...}
   public void _read(
         org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
         org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class FixedHolder
      implements org.omg.CORBA.portable.Streamable {
   public java.math.BigDecimal value;
   public FixedHolder() {}
   public FixedHolder(java.math.BigDecimal initial) {...}
   public void _read(
         org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
         org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
```

```
}
```

The Holder class for a user defined type *<foo>* is shown below:

```
// Java
final public class <foo>Holder
      implements org.omg.CORBA.portable.Streamable {
   public <foo> value;
   public <foo>Holder() {}
   public <foo>Holder(<foo> initial) {...}
   public void _read(
         org.omg.CORBA.portable.InputStream is) {...}
   public void _write(
         org.omg.CORBA.portable.OutputStream os) {...}
   public org.omg.CORBA.TypeCode _type() {...}
   }
```

1.4.1.5 Use of Java `null`

The Java **`null`** may only be used to represent the "null" object reference or value type. For example, a zero length string rather than **`null`** must be used to represent the empty string. Similarly for arrays.

1.4.2 Boolean

The IDL boolean constants **TRUE** and **FALSE** are mapped to the corresponding Java boolean literals **`true`** and **`false`**.

1.4.3 Character Types

IDL characters are 8-bit quantities representing elements of a character set while Java characters are 16-bit unsigned quantities representing Unicode characters. In order to enforce type-safety, the Java CORBA runtime asserts range validity of all Java **`chars`** mapped from IDL **chars** when parameters are marshaled during method invocation. If the **`char`** falls outside the range defined by the character set, a CORBA::DATA_CONVERSION exception shall be thrown.

The IDL **wchar** maps to the Java primitive type **`char`**. If the wchar falls outside the range defined by the character set, a CORBA::DATA_CONVERSION exception shall be thrown.

1.4.4 Octet

The IDL type **octet**, an 8-bit quantity, is mapped to the Java type **`byte`**.

1.4.5 String Types

The IDL **string**, both bounded and unbounded variants, are mapped to **`java.lang.String`**. Range checking for characters in the string as well as bounds checking of the string is done at marshal time. Character range violations cause a CORBA::DATA_CONVERSION exception to be raised. Bounds violations cause a CORBA::BAD_PARAM exception to be raised.

The IDL **wstring**, both bounded and unbounded variants, are mapped to **`java.lang.String`**. Bounds checking of the string is done at marshal time. Character range violations cause a CORBA::DATA_CONVERSION exception to be raised. Bounds violations cause a CORBA:: BAD_PARAM exception to be raised.

1.4.6 Integer Types

The integer types map as shown in Table 1-1 on page 1-6.

1.4.7 Floating Point Types

The IDL float and double map as shown in Table 1-1 on page 1-6.

1.4.8 Fixed Point Types

The IDL **fixed** type is mapped to the Java **`java.math.BigDecimal`** class. Range checking is done at marshal time. Size violations cause a CORBA::DATA_CONVERSION exception to be raised.

1.4.9 Future Long Double Types

There is no current support in Java for the IDL **long double** type. It is not clear at this point whether and when this type will be added either as a primitive type, or as a new package in **`java.math.*`**, possibly as **`java.math.BigFloat`**.

This is left for a future revision.

1.5 Helpers

All user defined IDL types have an additional "helper" Java class with the suffix **`Helper`** appended to the type name generated. (Note that in this context user defined includes IDL types that are defined in OMG specifications such as those for the Interface Repository and other OMG services.)

1.5.1 Helpers for Boxed Values

Although helper classes are generated for boxed value types, some of their specifics differ from the helpers for other user defined types. See Section 1.14, "Value Box Types," on page 1-46 for the details.

1.5.2 Helper Classes (except Boxed Values)

Several static methods needed to manipulate the type are supplied. These include **Any** insert and extract operations for the type, getting the repository id, getting the typecode, and reading and writing the type from and to a stream.

The helper class for a mapped IDL interface or abstract interface also have narrow operation(s) defined in the template below.

For any user defined, non-boxed value type IDL type, <***typename***>, the following is the Java code generated for the type.

```
// generated Java helper - non boxed value types

abstract public class <typename>Helper {
   public static void
      insert(org.omg.CORBA.Any a, <typename> t) {...}
   public static <typename> extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static <typename> read(
         org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
      org.omg.CORBA.portable.OutputStream os,<typename> val)
      {...}

   // only helpers for abstract interface
   public static <typename> narrow(
               java.lang.Object obj)
      {...}

   // only helpers for non-abstract interface with at
   // least one abstract base interface
   public static <typename> narrow(org.omg.CORBA.Object obj)
      {...}
   public static <typename> narrow(java.lang.Object obj)
      {...}

   // only helpers for non-abstract interface with
   // no abstract base interface
   public static <typename> narrow(
               org.omg.CORBA.Object obj)
      {...}

   // for each factory declaration in non abstract
   // value type
   public static <typename> <factoryname> (
               org.omg.CORBA.ORB orb
               [ "," <factoryarguments>] )
      {...}
```

```
}
```

1.5.2.1 Value type Factory Convenience Methods

For each factory declaration in a value type declaration, a corresponding static convenience method is generated in the helper class for the value type. The name of this method is the name of the factory.

This method takes an **orb** instance and all the arguments specified in the factory argument list. The implementation of each of these methods will locate a ***<typename>*****ValueFactory** (see Section 1.13.8, “Value Factory and Marshaling,” on page 1-45) and call the identically named method on the **ValueFactory** passing in the supplied arguments.

1.5.3 Examples

```
// IDL - named type
struct stfoo {long f1; string f2;};

// generated Java
abstract public class stfooHelper {
   public static void insert(
                        org.omg.CORBA.Any a,
                        stfoo t)
      {...}
   public static stfoo extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static stfoo read(
               org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
               org.omg.CORBA.portable.OutputStream os,
               stfoo val)
      {...}
}

// IDL - typedef sequence
typedef sequence <long> IntSeq;

// generated Java helper

public abstract class IntSeqHelper {
   public static void insert(
                              org.omg.CORBA.Any a,
                              int[] t)
   public static int[] extract(Any a){...}
   public static org.omg.CORBA.TypeCode type(){...}
   public static String id(){...}
```

```
    public static int[] read(
                org.omg.CORBA.portable.InputStream is)
        {...}
    public static void write(
                org.omg.CORBA.portable.OutputStream os,
                int[] val)
        {...}
}
```

1.6 Mapping for Constant

Constants are mapped differently depending upon the scope in which they appear.

1.6.1 Constants Within An Interface

Constants declared within an IDL interface are mapped to fields in either the Java signature interface for non-abstract or the sole Java interface for abstract (see Section 1.12, “Mapping for Interface,” on page 1-27) corresponding to the IDL interface.

1.6.1.1 Example

```
// IDL

module Example {
    interface Face {
    const long aLongerOne = -321;
    };
};

// generated Java

package Example;

public interface FaceOperations {
}

public interface Face extends FaceOperations,
                     org.omg.CORBA.Object,
                     org.omgCORBA.portable.IDLEntity {
    int aLongerOne = (int) (-321L)
}

// Helper and Holder classes omitted for simplicity
```

1.6.2 Constants Not Within An Interface

Constants not declared within an IDL interface are mapped to a **public interface** with the same name as the constant and containing a field, named **value**, that holds the contant's value. Note that the Java compiler will normally inline the value when the class is used in other Java code.

1.6.2.1 Example

```
// IDL

module Example {
    const long aLongOne = -123;
    };

package Example;
public interface aLongOne {
    int value = (int) (-123L);
}
```

1.7 Mapping for Enum

An IDL **enum** is mapped to a Java **class** which implements **IDLEntity** with the same name as the enum type which declares a value method, two static data members per label, an integer conversion method, and a protected constructor as follows:

```
// generated Java

public class <enum_name>
         implements org.omg.CORBA.portable.IDLEntity {

    // one pair for each label in the enum
    public static final int _<label> = <value>;
    public static final <enum_name> <label> =
        new <enum_name>(_<label>);

    public int value() {...}

    // get enum with specified value
    public static <enum_name> from_int(int value);

    // constructor
    protected <enum_name>(int) {...}
}
```

One of the members is a **public static final** that has the same name as the IDL enum label. The other has an underscore (_) prepended and is intended to be used in switch statements.

The value method returns the integer value. Values are assigned sequentially starting with 0. Note: there is no conflict with the **value()** method in Java even if there is a label named **value**.

There shall be only one instance of an enum. Since there is only one instance, equality tests will work correctly. For example, the default java.lang.Object implementation of **equals()** and **hash()** will automatically work correctly for an enum's singleton object.

The Java class for the enum has an additional method **from_int()**, which returns the enum with the specified value.

A helper class is also generated according to the normal rules, see Section 1.5, "Helpers," on page 1-13.

The holder class for the enum is also generated. Its name is the enum's mapped Java classname with **Holder** appended to it as follows:

```
public class <enum_name>Holder implements
      org.omg.CORBA.portable.Streamable {
   public <enum_name> value;
   public <enum_name>Holder() {}
   public <enum_name>Holder(<enum_name> initial) {...}
   public void _read(org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
            org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}
```

1.7.1 Example

```
// IDL
enum EnumType {first, second, third, fourth, fifth};

// generated Java

public class EnumType
         implements org.omg.CORBA.portable.IDLEntity {

   public static final int _first = 0;
   public static final EnumType first =
            new EnumType(_first);

   public static final int _second = 1;
   public static final EnumType second =
            new EnumType(_second);

   public static final int _third = 2;
   public static final EnumType third =
```

```
                new EnumType(_third);

    public static final int _fourth = 3;
    public static final EnumType fourth =
                new EnumType(_fourth);

    public static final int _fifth = 4;
    public static final EnumType fifth =
                new EnumType(_fifth);

    public int value() {...}
    public static EnumType from_int(int value) {...};

    // constructor
    protected EnumType(int) {...}

    };

// generated Java helper

abstract public class EnumTypeHelper {
    public static void insert(
                    org.omg.CORBA.Any a, EnumType t)
        {...}
    public static EnumType extract(Any a) {...}
    public static org.omg.CORBA.TypeCode type() {...}
    public static String id() {...}
    public static EnumType read(
                    org.omg.CORBA.portable.InputStream is)
        {...}
    public static void write(
                    org.omg.CORBA.portable.OutputStream os,
                    EnumType val)
        {...}
}

final public class EnumTypeHolder
        implements org.omg.CORBA.portable.Streamable {
    public EnumType value;
    public EnumTypeHolder() {}
    public EnumTypeHolder(EnumType initial) {...}
    public void _read(
                org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                org.omg.CORBA.portable.OutputStream os)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
    }
```

1.8 Mapping for Struct

An IDL **struct** is mapped to a final Java class with the same name and which provides instance variables for the fields in IDL member ordering, a constructor for all values, and which implements **IDLEntity**. A null constructor is also provided so that the fields can be filled in later.

A helper class is also generated according to the normal rules, see Section 1.5, “Helpers,” on page 1-13.

The holder class for the struct is also generated. Its name is the struct’s mapped Java classname with **Holder** appended to it as follows:

```
final public class <class>Holder implements
        org.omg.CORBA.portable.Streamable {
    public <class> value;
    public <class>Holder() {}
    public <class>Holder(<class> initial) {...}
    public void _read(
                    org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream os)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

1.8.1 Example

```
// IDL
struct StructType {
    long field1;
    string field2;
};

// generated Java
final public class StructType
        implements org.omg.CORBA.portable.IDLEntity {
    // instance variables
    public int field1;
    public String field2;
    // constructors
    public StructType() {}
    public StructType(int field1, String field2)
        {...}
    }

final public class StructTypeHolder
        implements org.omg.CORBA.portable.Streamable {
    public StructType value;
```

```
    public StructTypeHolder() {}
    public StructTypeHolder(StructType initial) {...}
    public void _read(
                    org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream os)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

abstract public class StructTypeHelper {
    public static void
        insert(org.omg.CORBA.Any a, StructType t) {...}
    public static StructType extract(Any a) {...}
    public static org.omg.CORBA.TypeCode type() {...}
    public static String id() {...}
    public static StructType read(
                    org.omg.CORBA.portable.InputStream is)
        {...}
    public static void write(
                    org.omg.CORBA.portable.OutputStream os,
                    StructType val)
        {...}
}
```

1.9 Mapping for Union

An IDL **union** is mapped to a final Java class with the same name, which implements **IDLEntity** and has:

- a default constructor
- an accessor method for the discriminator, named **discriminator()**
- an accessor method for each branch
- a modifier method for branch
- a modifier method for each branch which has more than one case label.
- a modifier method for the branch corresponding to the default label if present
- a default modifier method if needed

The normal name conflict resolution rule is used (prepend an "_") for the discriminator if there is a name clash with the mapped uniontype name or any of the field names.

The branch accessor and modifier methods are overloaded and named after the branch. Accessor methods shall raise the CORBA::BAD_OPERATION system exception if the expected branch has not been set.

If there is more than one case label corresponding to a branch, the simple modifier method for that branch sets the discriminant to the value of the first case label. In addition, an extra modifier method which takes an explicit discriminator parameter is generated.

If the branch corresponds to the **default** case label, then the simple modifier for that branch sets the discriminant to the first available default value starting from a 0 index of the discriminant type. In addition, an extra modifier which takes an explicit discriminator parameter is generated.

It is illegal to specify a union with a default case label if the set of case labels completely covers the possible values for the discriminant. It is the responsibility of the Java code generator (e.g., the IDL complier, or other tool) to detect this situation and refuse to generate illegal code.

Two default modifier methods, both named **__default()**, are generated if there is no explicit default case label, and the set of case labels does not completely cover the possible values of the discriminant. The simple method taking no arguments and returning void sets the discriminant to the first available default value starting from a 0 index of the discriminant type. The second method takes a discriminator as a parameter and returns void. Both of these of methods shall leave the union with a discriminator value set, and the value member uninitialized.

A helper class is also generated according to the normal rules, see Section 1.5, “Helpers,” on page 1-13.

The holder class for the union is also generated. Its name is the union’s mapped Java classname with **Holder** appended to it as follows:

```
final public class <union_class>Holder
        implements org.omg.CORBA.portable.Streamable {
   public <union_class> value;
   public <union_class>Holder() {}
   public <union_class>Holder(<union_class> initial) {...}
   public void _read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                  org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}
```

1.9.1 Example

// IDL - EnumType from Section 1.7.1, “Example,” on page 1-18

```
union UnionType switch (EnumType) {
        case first: long win;
        case second: short place;
        case third:
        case fourth: octet show;
        default:    boolean other;
    };

// generated Java

final public class UnionType
                implements org.omg.CORBA.portable.IDLEntity {

    // constructor
    public UnionType() {....}

    // discriminator accessor
    public EnumType discriminator() {....}

    // win
    public int win() {....}
    public void win(int value) {....}

    // place
    public short place() {....}
    public void place(short value) {....}

    // show
    public byte show() {....}
    public void show(byte value) {....}
    public void show(EnumType discriminator, byte
value){....}

    // other
    public boolean other() {....}
    public void other(boolean value) {....}
    public void other(EnumType discriminator, boolean value)
{....}
}

final public class UnionTypeHolder
        implements org.omg.CORBA.portable.Streamable {
    public UnionType value;
    public UnionTypeHolder() {}
    public UnionTypeHolder(UnionType initial) {...}
    public void _read(
                org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                org.omg.CORBA.portable.OutputStream os)
        {...}
```

```
   public org.omg.CORBA.TypeCode _type() {...}

}
abstract public class UnionTypeHelper {
   public static void
      insert(org.omg.CORBA.Any a, UnionType t) {...}
   public static UnionType extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static UnionType read(
                  org.omg.CORBA.portable.InputStream is)
      {...}

   public static void write(
                  org.omg.CORBA.portable.OutputStream os,
                  UnionType val)
      {...}
```

1.10 Mapping for Sequence

An IDL **sequence** is mapped to a Java array with the same name. In the mapping, everywhere the sequence type is needed, an array of the mapped type of the sequence element is used. Bounds checking shall be done on bounded sequences when they are marshaled as parameters to IDL operations, and an IDL CORBA::MARSHAL is raised if necessary.

A helper class is also generated according to the normal rules, see Section 1.5, “Helpers,” on page 1-13.

The holder class for the sequence is also generated. Its name is the sequence’s mapped Java classname with **Holder** appended to it as follows:

```
final public class <sequence_class>Holder {
   public <sequence_element_type>[] value;
   public <sequence_class>Holder() {};
   public <sequence_class>Holder(
                  <sequence_element_type>[] initial)
      {...};
   public void _read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                  org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}
```

1.10.1 Example

// IDL

typedef sequence< long > UnboundedData;
typedef sequence< long, 42 > BoundedData;

```
// generated Java

final public class UnboundedDataHolder
         implements org.omg.CORBA.portable.Streamable {
   public int[] value;
   public UnboundedDataHolder() {};
   public UnboundedDataHolder(int[] initial) {...};
   public void _read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                  org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

abstract public class UnBoundedDataHelper {
   public static void
      insert(org.omg.CORBA.Any a, int[] t) {...}
   public static int[] extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static int[] read(
               org.omg.CORBA.portable.InputStream istream)
      {...}
   public static void write(
               org.omg.CORBA.portable.OutputStream ostream,
               int[] val)
      {...}
}

final public class BoundedDataHolder
         implements org.omg.CORBA.portable.Streamable {
   public int[] value;
   public BoundedDataHolder() {};
   public BoundedDataHolder(int[] initial) {...};
   public void _read(org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(org.omg.CORBA.portable.OutputStream
os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

abstract public class BoundedDataHelper {
   public static void
      insert(org.omg.CORBA.Any a, int[] t) {...}
   public static int[] extract(Any a) {...}
```

```
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static int[] read(
               org.omg.CORBA.portable.InputStream istream)
      {...}
   public static void write(
               org.omg.CORBA.portable.OutputStream ostream,
               int[] val)
      {...}
}
```

1.11 Mapping for Array

An IDL array is mapped the same way as an IDL bounded sequence. In the mapping, everywhere the array type is needed, an array of the mapped type of the array element is used. In Java, the natural Java subscripting operator is applied to the mapped array. The bounds for the array are checked when the array is marshaled as an argument to an IDL operation and a CORBA::MARSHAL exception is raised if a bounds violation occurs. The length of the array can be made available in Java, by bounding the array with an IDL constant, which will be mapped as per the rules for constants.

The holder class for the array is also generated. Its name is the array's mapped Java classname with **Holder** appended to it as follows:

```
final public class <array_class>Holder
      implements org.omg.CORBA.portable.Streamable {
   public <array_element_type>[] value;
   public <array_class>Holder() {}
   public <array_class>Holder(
                  <array_element_type>[] initial) {...}
   public void _read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                  org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}
```

1.11.1 Example

```
// IDL

const long ArrayBound = 42;
typedef long larray[ArrayBound];

// generated Java

public interface ArrayBound {
```

```
   int value = (int) 42;
}

final public class larrayHolder
      implements org.omg.CORBA.portable.Streamable {
   public int[] value;
   public larrayHolder() {}
   public larrayHolder(int[] initial) {...}
   public void _read(
               org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
               org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

abstract public class larrayHelper {
   public static void
      insert(org.omg.CORBA.Any a, int[] t) {...}
   public static int[] extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static int[] read(
            org.omg.CORBA.portable.InputStream istream)
      {...}
   public static void write(
            org.omg.CORBA.portable.OutputStream ostream,
            int[] val)
      {...}
}
```

1.12 Mapping for Interface

1.12.1 Basics

A non abstract IDL **interface** is mapped to two public Java interfaces: a *signature interface* and an *operations interface*. The signature interface, which extends **IDLEntity**, has the same name as the IDL interface name and is used as the signature type in method declarations when interfaces of the specified type are used in other interfaces. The operations interface has the same name as the IDL interface with the suffix **Operations** appended to the end and is used in the server-side mapping and as a mechanism for providing optimized calls for collocated client and servers.

A helper class is also generated according to the normal rules, see Section 1.5, “Helpers,” on page 1-13.

The Java operations interface contains the mapped operation signatures. The Java signature interface extends the operations interface, the (mapped) base **org.omg.CORBA.Object**, as well as **org.omg.portable.IDLEntity**. Methods can be invoked on the signature interface. Interface inheritance expressed in IDL is reflected in both the Java signature interface and operations interface hierarchies.

The helper class holds a static narrow method that allows an **org.omg.CORBA.Object** to be narrowed to the object reference of a more specific type. The IDL exception CORBA::BAD_PARAM is thrown if the narrow fails because the object reference does not support the requested type. A different system exception is raised to indicate other kinds of errors. Trying to narrow a **null** will always succeed with a return value of **null**.

There are no special “nil” object references. Java **null** can be passed freely wherever an object reference is expected.

Attributes are mapped to a pair of Java accessor and modifier methods. These methods have the same name as the IDL attribute and are overloaded. There is no modifier method for IDL **readonly** attributes.

The holder class for the interface is also generated. Its name is the interface’s mapped Java classname with **Holder** appended to it as follows:

```
final public class <interface_class>Holder
      implements org.omg.CORBA.portable.Streamable {
   public <interface_class> value;
   public <interface_class>Holder() {}
   public <interface_class>Holder(
                  <interface_class> initial) {
      value = initial;
   public void _read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                  org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}
```

Abstract Interfaces

An IDL **abstract interface** is mapped to a single public Java interface with the same name as the IDL interface. The mapping rules are similar to the rules for generating the Java operations interface for a non-abstract IDL interface. However this interface also serves as the signature interface, and hence extends **org.omg.CORBA.portable.IDLEntity**. The mapped Java interface, has the same name as the IDL interface name and is also used as the signature type in method declarations when interfaces of the specified type are used in other interfaces. It contains the methods which are the mapped operations signatures.

A holder class is generated as for non-abstract interfaces.

A helper class is also generated according to the normal rules, see Section 1.5, "Helpers," on page 1-13.

CORBA::AbstractBase is mapped to `java.lang.Object`.

1.12.1.1 Example

```
// IDL

module Example {
    interface Marker {
    };
    abstract interface Base {
        void baseOp();
    };
    interface Extended: Base, Marker {
        long method (in long arg) raises (e);
        attribute long assignable;
        readonly attribute long nonassignable;
    }
}

// generated Java

package Example;

public interface MarkerOperations {
}

public interface Base extends
                org.omg.CORBA.portable.IDLEntity {
   void baseOp();
}

public interface ExtendedOperations extends
                            Base, MarkerOperations {
   int method(int arg)
      throws Example.e;
   int assignable();
   void assignable(int i);
   int nonassignable();
}

public interface Marker extends MarkerOperations,
                org.omg.CORBA.Object,
                org.omg.CORBA.portable.IDLEntity {
}
```

```
public interface Extended extends ExtendedOperations,
               Marker,
               org.omg.CORBA.portable.IDLEntity {
}

abstract public class ExtendedHelper {
   public static void
      insert(org.omg.CORBA.Any a, Extended t) {...}
   public static Extended extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static Extended read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
                  org.omg.CORBA.portable.OutputStream os,
                  Extended val)
      {...}
   public static Extended narrow(org.omg.CORBA.Object obj)
      {...}
   public static Extended narrow(java.lang.Object obj)
      {...}
}

abstract public class BaseHelper {
   public static void
      insert(org.omg.CORBA.Any a, Base t) {...}
   public static Base extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static Base read(
                     org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
                     org.omg.CORBA.portable.OutputStream os,
                     Base val)
      {...}
   public static Base narrow(java.lang.Object obj)
      {...}
}

abstract public class MarkerHelper{
   public static void
      insert(org.omg.CORBA.Any a, Marker t) {...}
   public static Marker extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static Marker read(
                     org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
```

```
                        org.omg.CORBA.portable.OutputStream os,
                        Marker val)
      {...}
   public static Marker narrow(org.omg.CORBA.Object obj)
      {...}
}

final public class ExtendedHolder
      implements org.omg.CORBA.portable.Streamable {
   public Extended value;
   public ExtendedHolder() {}
   public ExtendedHolder(Extended initial) {...}
   public void _read(
                        org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                        org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class BaseHolder
      implements org.omg.CORBA.portable.Streamable {
   public Base value;
   public BaseHolder() {}
   public BaseHolder(Base initial) {...}
   public void _read(
                     org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                     org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

final public class MarkerHolder
      implements org.omg.CORBA.portable.Streamable {
   public Marker value;
   public MarkerHolder() {}
   public MarkerHolder(Marker initial) {...}
   public void _read(
                     org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                     org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}
```

1.12.2 Parameter Passing Modes

IDL **in** parameters, which implement call-by-value semantics, are mapped to normal Java parameters. The results of IDL operations are returned as the result of the corresponding Java method.

IDL **out** and **inout** parameters, which implement call-by-result and call-by-value/result semantics, cannot be mapped directly into the Java parameter passing mechanism. This mapping defines additional holder classes for all the IDL basic and user-defined types which are used to implement these parameter modes in Java. The client supplies an instance of the appropriate holder Java class that is passed (by value) for each IDL out or inout parameter. The contents of the holder instance (but not the instance itself) are modified by the invocation, and the client uses the (possibly) changed contents after the invocation returns.

- For IDL **in** parameters that are not valuetypes:
  - Java objects passed for non-valuetype IDL **in** parameters are created and owned by the caller. With the exception of value types, the callee must not modify **in** parameters or retain a reference to the **in** parameter beyond the duration of the call. Violation of these rules can result in unpredictable behavior.
- For IDL **in** parameters that are valuetypes:
  - Java objects passed for valuetype IDL **in** parameters are created by the caller and a copy is passed to the callee. The callee may modify or retain a reference to the copy beyond the duration of the call.
- Java objects returned as IDL **out** or return parameters are created and owned by the callee. Ownership of such objects transfers to the caller upon completion of the call. The callee must not retain a reference to such objects beyond the duration of the call. Violation of these rules can result in unpredictable behavior.

- IDL **inout** parameters have the above **in** semantics for the **in** value, and have the above **out** semantics for the **out** value.
- The above rules do not apply to Java primitive types and immutable Java objects. Currently, the only immutable Java class used by this mapping is `java.lang.String`.

1.12.2.1 Example

```
// IDL

module Example {
```

```
    interface Modes {
        long operation(in long inArg,
                        out long outArg,
                        inout long inoutArg);
    };
};

// Generated Java

package Example;

public interface ModesOperations {
    int operation(int inArg,
            org.omg.CORBA.IntHolder outArg,
            org.omg.CORBA.IntHolder inoutArg);
}

public interface Modes extends ModesOperations,
org.omg.CORBA.Object,
        org.omg.CORBA.portable.IDLEntity {
}

final public class ModesHolder
        implements org.omg.CORBA.portable.Streamable {
    public Modes value;
    public ModesHolder() {}
    public ModesHolder(Modes initial) {...}
    public void _read(
                org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                org.omg.CORBA.portable.OutputStream os)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
    }

abstract public class ModesHelper {
    public static void
        insert(org.omg.CORBA.Any a, Modes t) {...}
    public static Modes extract(Any a) {...}
    public static org.omg.CORBA.TypeCode type() {...}
    public static String id() {...}
    public static Modes read(
                    org.omg.CORBA.portable.InputStream is)
        {...}
    public static void write(
                    org.omg.CORBA.portable.OutputStream os,
                    Modes val)
        {...}
    public static Modes narrow(java.lang.Object obj)
        {...}
}
```

In the above, the result comes back as an ordinary result and the actual in parameters only is an ordinary value. But for the out and inout parameters, an appropriate holder must be constructed. A typical use case might look as follows:

```
// user Java code

// select a target object
Example.Modes target = ...;

// get the in actual value
int inArg = 57;

// prepare to receive out
IntHolder outHolder = new IntHolder();

// set up the in side of the inout
IntHolder inoutHolder = new IntHolder(131);

// make the invocation
int result =target.operation(inArg, outHolder, inoutHolder);

// use the value of the outHolder
... outHolder.value ...

// use the value of the inoutHolder
... inoutHolder.value ...
```

Before the invocation, the input value of the inout parameter must be set in the holder instance that will be the actual parameter. The inout holder can be filled in either by constructing a new holder from a value, or by assigning to the value of an existing holder of the appropriate type. After the invocation, the client uses the outHolder.value to access the value of the out parameter, and the inoutHolder.value to access the output value of the inout parameter. The return result of the IDL operation is available as the result of the invocation.

1.12.3 Context Arguments to Operations

If an operation in an IDL specification has a context specification, then an **org.omg.CORBA.Context** input parameter (see Section 1.19.7, “Context,” on page 1-67) is appended following the operation-specific arguments, to the argument list for an invocation.

1.13 Mapping for Value Type

1.13.1 Java Interfaces Used For Value Types

This section describes several Java interfaces which are used (and required) as part of the Java mapping for IDL value types.

1.13.1.1 ValueBase Interface

```
package org.omg.CORBA.portable;

public interface ValueBase extends IDLEntity {
   String[] _truncatable_ids();
}

package org.omg.CORBA;

abstract public class ValueBaseHelper {
   public static void
      insert(org.omg.CORBA.Any a, java.io.Serializable t)
         {...}
   public static java.io.Serializable
      extract(org.omg.CORBA.Any a)
         {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static java.io.Serializable read(
                     org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
                     org.omg.CORBA.portable.OutputStream os,
                     java.io.Serializable val)
      {...}
}

final public class ValueBaseHolder
      implements org.omg.CORBA.portable.Streamable {
   public java.io.Serializable value;
   public ValueBaseHolder() {}
   public ValueBaseHolder(java.io.Serializable initial)
      {...}
   public void _read(org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                     org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
   }
```

All value types implement ValueBase either directly (see Section 1.14.2, “Boxed Primitive Types,” on page 1-46), or indirectly by implementing either the **StreamableValue** or **CustomValue** interface (see below).

1.13.1.2 StreamableValue Interface

```
package org.omg.CORBA.portable;
```

```
public interface StreamableValue extends
                        Streamable, ValueBase {}
```

All non-boxed IDL valuetypes that are not custom marshaled implement the **StreamableValue** interface.

1.13.1.3 CustomMarshal Interface

```
package org.omg.CORBA;

public interface CustomMarshal {
   public void marshal(org.omg.CORBA.DataOutputStream os);
   public void unmarshal(org.omg.CORBA.DataInputStream is);
}
```

Implementers of custom marshaled values implement the **CustomMarshal** interface to provide custom marshaling.

1.13.1.4 CustomValue Interface

```
package org.omg.CORBA.portable;

public interface CustomValue extends ValueBase,
               org.omg.CORBA.CustomMarshal {
}
```

All custom value types generated from IDL implement the **CustomValue** interface.

1.13.1.5 ValueFactory Interface

```
package org.omg.CORBA.portable;

public interface ValueFactory {
   java.io.Serializable read_value(
               org.omg.CORBA_2_3.portable.InputStream is);
}
```

The **ValueFactory** interface is the native mapping for the IDL type **CORBA::ValueFactory**. The **read_value()** method is called by the ORB runtime while in the process of unmarshaling a value type. A user implements this method as part of implementing a type specific value factory. In the implementation, the user calls **is.read_value(java.io.Serializable)** with an uninitialized valuetype to use for unmarshaling. The value returned by the stream is the same value passed in, with all the data unmarshaled.

1.13.2 Basics For Stateful Value Types

A concrete value type (i.e., one that is not declared as abstract) is mapped to an abstract Java class with the same name, and a factory Java interface with the suffix "**ValueFactory**" appended to the value type name. In addition, a helper class with the suffix "**Helper**" appended to the value type name and a holder class with the suffix "**Holder**" appended to the value type name is generated.

The specification of the generated holder class is as follows:

```
public final class <typename>Holder implements
      org.omg.CORBA.portable.Streamable {
   public <typename> value;
   public <typename>Holder () {}
   public <typename>Holder (final <typename> initial) {
      value = initial;
   }
   public void _read (
            final org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
            final org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type () {...}
}
```

The value type's mapped Java abstract class contains instance variables that correspond to the fields in the state definition in the IDL declaration. The order and name of the Java instance variables are the same as the corresponding IDL state fields. Fields that are identified as **public** in the IDL are mapped to **public** instance variables. Fields that are identified as **private** in the IDL are mapped to **protected** instance variables in the mapped Java class.

The Java class for the value type extends either **org.omg.CORBA.portable.CustomValue** or **org.omg.CORBA.portable.StreamableValue**, depending on whether it is declared as custom in IDL or not, respectively.

The generated Java class shall provide an implementation of the **ValueBase** interface for this value type. For value types that are streamable (i.e., non-custom), the generated Java class also provides an implementation for the **org.omg.CORBA.portable.Streamable** interface.

The value type's generated value factory interface extends **org.omg.CORBA.portable.ValueFactory** and contains one method corresponding to each factory declared in the IDL. The name of the method is the same as the name of the factory, and the factory arguments are mapped in the same way as **in** parameters are for IDL operations.

The implementor provides a factory class with implementations for the methods in the generated value factory interface. When no factories are declared in IDL, then the value type's value factory is eliminated from the mapping and the implementor simply implements **org.omg.CORBA.portable.ValueFactory** to provide the method body for **read_value()**.

The mapped Java class contains abstract method definitions that correspond to the operations and attributes defined on the value type in IDL.

An implementor of the value type extends the generated Java class to provide implementation for the operations and attributes declared in the IDL, including those for any derived or supported value types or interfaces.

1.13.2.1 Inheritance From Value Types

The inheritance scheme and specifics of the mapped class depend upon the inheritance and implementation characteristics of the value type and are described in the following subsections.

Value types that do not inherit from other values or interfaces:

For non custom values, the generated Java class also implements the **StreamableValue** interface and provides appropriate implementation to marshal the state of the object. For custom values, the generated class extends **CustomValue** but does not provide an implementation for the **CustomMarshal** methods.

Inheritance from other stateful values

The generated Java class extends the Java class to which the inherited value type is mapped. If the valuetype is custom, but its base is not, then the generated Java class also implements the **CustomValue** interface.

Inheritance from abstract values

The generated Java class implements the Java interface to which the inherited abstract value is mapped (see Section 1.13.3, "Abstract Value Types," on page 1-39).

Supported interfaces

The Java class implements the Operations Java interface of all the interfaces, if any, that it supports. (Note that the operations interface for abstract interfaces does not have the "Operations" suffix, see "Abstract Interfaces" on page 1-28). It also implements the appropriate interface, either **StreamableValue** or **CustomValue**, as per the rules stated in "Value types that do not inherit from other values or interfaces:" on page 1-38. The implementation of the supported interfaces of the value type shall use the tie mechanism, to tie to the value type implementation.

1.13.3 Abstract Value Types

An abstract value type maps to a Java interface that extends **ValueBase** and contains all the operations and attributes specified in the IDL, mapped using the normal rules for mapping operations and attributes.

Abstract value types cannot be implemented directly. They must only be inherited by other stateful value types or abstract value types.

1.13.4 CORBA::ValueBase

CORBA::ValueBase is mapped to **java.io.Serializable**.

The **get_value_def()** operation is not mapped to any of the classes associated with a value type in Java. Instead it appears as an operation on the ORB pseudo object in Java (see "public org.omg.CORBA.Object get_value_def(String repId)" in Section 1.19.10, "ORB," on page 1-75.

1.13.5 Example A

```
// IDL

typedef sequence<unsigned long> WeightSeq;

module ExampleA {
    valuetype WeightedBinaryTree {
        private long weight;
        private WeightedBinaryTree left;
        private WeightedBinaryTree right;
        factory createWBT(in long w);
        WeightSeq preOrder();
        WeightSeq postOrder();
    };
};

// generated Java

package ExampleA;

public abstract class WeightedBinaryTree
    implements org.omg.CORBA.portable.StreamableValue {

    // instance variables and IDL operations
    protected int weight;
    protected WeightedBinaryTree left;
    protected WeightedBinaryTree right;
    public abstract int[] preOrder();
    public abstract int[] postOrder();

    // from ValueBase
```

```
   public String[] _truncatable_ids() {...}

   // from Streamable
   public void _read(
                 org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                 org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

public interface WeightedBinaryTreeValueFactory extends
      org.omg.CORBA.portable.ValueFactory {
   WeightedBinaryTree createWBT(int weight) {...}
}

abstract public class WeightedBinaryTreeHelper {
   public static void insert(
              org.omg.CORBA.Any a, WeightedBinaryTree t)
      {...}
   public static WeightedBinaryTree extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static WeightedBinaryTree read(
              org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
              org.omg.CORBA.portable.OutputStream os,
              WeightedBinaryTree val)
      {...}

   // for factory
      public static WeightedBinaryTree createWBT(
              org.omg.CORBA.ORB orb,
              int weight)
      {...}
}

// Holder class
final public class WeightedBinaryTreeHolder
              implements org.omg.CORBA.portable.Streamable {
   public WeightedBinaryTree value;
   public WeightedBinaryTreeHolder() {}
   public WeightedBinaryTreeHolder(
                 WeightedBinaryTreeHolder initial)
      {...}
   public void _read(
           org.omg.CORBA.portable.InputStream is)
      {
      // read state information using the wire format
```

```
        // and construct value
        ...
        }
    public void _write(
            org.omg.CORBA.portable.OutputStream os) {
        // write state information using the wire format
        ...
        }
    public org.omg.CORBA.TypeCode _type() {...}
}
```

1.13.6 Example B

```
// IDL

module ExampleB {
    interface Printer {
        typedef sequence<unsigned long> ULongSeq;
        void print(in ULongSeq data);
        };
    valuetype WeightedBinaryTree supports Printer {
        private long weight;
        public WeightedBinaryTree left;
        public WeightedBinaryTree right;
        ULongSeq preOrder();
        ULongSeq postOrder();
        };
    };

// generated Java

package ExampleB;

final public class ULongSeqHolder
        implements org.omg.CORBA.portable.Streamable {
    public int[] value;
    public ULongSeqHolder() {}
    public ULongSeqHolder(int[] initial) {...}
    public void _read(
                org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                org.omg.CORBA.portable.OutputStream os)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
    }

public interface PrinterOperations {
    void print (int[] data);
}
```

```
public interface Printer extends
         PrinterOperations,
         org.omg.CORBA.Object,
         org.omg.CORBA.portable.IDLEntity {
}

abstract public class PrinterHelper {
   public static void
      insert(org.omg.CORBA.Any a, Printer t) {...}
   public static Printer extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static Printer read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
                  org.omg.CORBA.portable.OutputStream os,
                  Printer val)
      {...}
   public static Printer narrow(
                  org.omg.CORBA.Object obj)
      {...}
}

final public class PrinterHolder implements
                     org.omg.CORBA.portable.Streamable {
   public Printer value;
   public PrinterHolder() {}
   public PrinterHolder(Printer initial) {...}
   public void _read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                  org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

public abstract class WeightedBinaryTree implements
      ExampleB.PrinterOperations,
      org.omg.CORBA.portable.StreamableValue {

   // instance variables and IDL operations
   protected int weight;
   public WeightedBinaryTree left;
   public WeightedBinaryTree right;
   public int[] preOrder() {...}
   public int[] postOrder() {...}
   public print(int[] data) {...}

   // from ValueBase
```

```
    public String[] _truncatable_ids();

    // from Streamable
    public void _read(
                    org.omg.CORBA.portable.InputStream is);
    public void _write(
                    org.omg.CORBA.portable.OutputStream os);
    public org.omg.CORBA.TypeCode _type();
}

abstract public class WeightedBinaryTreeHelper {
    public static void insert(
                org.omg.CORBA.Any a, WeightedBinaryTree t)
        {...}
    public static WeightedBinaryTree extract(Any a) {...}
    public static org.omg.CORBA.TypeCode type() {...}
    public static String id() {...}
    public static WeightedBinaryTree read(
                org.omg.CORBA.portable.InputStream is)
        {...}
    public static void write(
                org.omg.CORBA.portable.OutputStream os,
                WeightedBinaryTree val)
        {...}
}

final public class WeightedBinaryTreeHolder implements
        org.omg.CORBA.portable.Streamable {
    public WeightedBinaryTree value;
    public WeightedBinaryTreeHolder() {}
    public WeightedBinaryTreeHolder(
                    WeightedBinaryTree initial)
        {...}
    public void _read(
                    org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream os)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

1.13.7 Parameter Passing Modes

If the formal parameter in the signature of an operation is a value type, then the actual parameter is passed by value. If the formal parameter type of an operation is an interface, then the actual parameter is passed by reference (i.e., it must be transformed to the mapped Java interface before being passed).

IDL value type **in** parameters are passed as the mapped Java class as defined above.

IDL value type **out** and **inout** parameters are passed using the Holder classes.

1.13.7.1 Example

```
// IDL - extended the above Example B

module ExampleB {

    interface Target {
        WeightedBinaryTree operation(in WeightedBinaryTree inArg,
                                     out WeightedBinaryTree outArg,
                                     inout WeightedBinaryTree inoutArg);
        };

    };
```

```
// generated Java code

package ExampleB;

public interface TargetOperations {
   WeightedBinaryTree operation(
                     WeightedBinaryTree inArg,
                     WeightedBinaryTreeHolder outArg,
                     WeightedBinaryTreeHolder inoutArg);

public interface Target extends
            TargetOperations,
            org.omgCORBA.Object,
            org.omg.CORBA.portable.IDLEntity {
}

abstract public class TargetHelper {
   public static void
      insert(org.omg.CORBA.Any a, Target t) {...}
   public static Target extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static Target read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
                  org.omg.CORBA.portable.OutputStream os,
                  Target val)
      {...}
   public static Target narrow(org.omg.CORBA.Object obj)
      {...}
}
final public class TargetHolder
      implements org.omg.CORBA.portable.Streamable {
```

```
public Target value;
public TargetHolder() {}
public TargetHolder(Target initial) {...}
public void _read(
            org.omg.CORBA.portable.InputStream is)
   {...}
public void _write(
            org.omg.CORBA.portable.OutputStream os) {...}
public org.omg.CORBA.TypeCode _type() {...}
}
```

1.13.8 Value Factory and Marshaling

Marshaling Java value instances is straightforward, but unmarshaling value instances is somewhat problematic. In Java there is no *a priori* relationship between the RepositoryID encoded in the stream and the class name of the actual Java class that implements the value. However, in practice we would expect that there will be a one-to-one relationship between the RepositoryID and the fully scoped name of the value type. However the RepositoryID may have an arbitrary prefix prepended to it, or be completely arbitrary.

The following algorithm will be followed by the ORB:

- Look up the value factory in the RepositoryID to value factory map.
- If this is not successful and the RepositoryId is a standard repository id that starts with “IDL:”, then attempt to generate the value factory class name to use by stripping off the “IDL:” header and “:<major>.<minor>” version information trailer, and replacing the “/”s that separate the module names with “.”s and appending “DefaultFactory.”
- If this is not successful and the RepositoryId is a standard repository id that starts with “RMI:” then attempt to generate the value factory class name from the RepositoryID by stripping off the “RMI::” header and the “:<hashcode> : <suid>” trailer, applying all the necessary conversions (see the *Java to IDL Language Mapping* document). If this class exists and implements **`org.omg.CORBA.portable.IDLEntity`**, then attempt to generate the value factory class name by appending “DefaultFactory”; otherwise, use the **`ValueHandler`** interface to read in the value (see the *Java to IDL Language Mapping* document, Section 1.5.1.3, “ValueHandler,” on page 1-40).
- If this is not successful, then raise the MARSHAL exception.

The IDL native type **ValueFactory** is mapped in Java to **`org.omg.CORBA.portable.ValueFactory`**.

A null is returned when **`register_value_factory()`** is called and no previous RepositoryId was registered.

As usual, it is a tools issue, as to how RepositoryIDs are registered with classes. It is our assumption that in the vast majority of times, the above default implicit registration policies will be adequate. A tool is free to arrange to have the ORB’s

register_value_factory() explicitly called if it wishes to explicitly register a particular Value Factory with some RepositoryID. For example, this could be done by an “installer” in a server, by pre-loading the ORB runtime, etc.

1.14 Value Box Types

The rules for mapping value box types are specified in this section. There are two general cases to consider: value boxes that are mapped to Java primitive types, and those that are mapped to Java classes.

Holder classes are generated for the value box types in the same way as for other value types. Helper classes are also generated, however they have a somewhat different structure and inheritance hierarchy than helpers generated for other value types.

1.14.1 Generic BoxedValueHelper Interface

Concrete helper classes for boxed values are generated. They all implement the following Java interface which serves as a base for boxed value helpers.

```
package org.omg.CORBA.portable;

public interface BoxedValueHelper {
   java.io.Serializable read_value(
                  org.omg.CORBA.portable.InputStream is);
   void write_value(
               org.omg.CORBA.portable.OutputStream os,
               java.io.Serializable value);
   java.lang.String get_id();
   }
```

1.14.2 Boxed Primitive Types

If the value box IDL type maps to a Java primitive (e.g., **float**, **long**, **char**, **wchar**, **boolean**, **octet**), then the value box type is mapped to a Java class whose name is the same as the IDL value type. The class has a public data member named `value`, and has the appropriate Java type. The holder and helper class are also generated.

```
// IDL
valuetype <box_name> <primitive_type>;

// generated Java

 public class <box_name> implements
                    org.omg.CORBA.portable.ValueBase {
   public <mapped_primitive_Java_type> value;
   public <box_name>(<mapped_primitive_Java_type> initial)
      { value = initial; }
```

```
    private static String[] _ids = { <box_name>Helper.id(); }
    public String[] _truncatable_ids() { return _ids;}
}

final public class <box_name>Holder
         implements org.omg.CORBA.portable.Streamable {
    public <mapped_primitive_Java_type> value;
    public <box_name>Holder() {}
    public <box_name>Holder(<box_name> initial) {...}
    public void _read(
                    org.omg.CORBA.portable.InputStream is)
       {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream os)
       {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

public class <box_name>Helper
         implements org.omg.CORBA.portable.BoxedValueHelper
{
    public static void insert(
                    org.omg.CORBA.Any a, <box_name> t)
       {...}
    public static <box_name> extract(Any a) {...}
    public static org.omg.CORBA.TypeCode type() {...}
    public static String id() {...}
    public static <box_name> read(
                    org.omg.CORBA.portable.InputStream is)
       {...}
    public static void write(
                    org.omg.CORBA.portable.OutputStream os,
                    <box_name> val)
       {...}
    public java.io.Serializable read_value(
                 org.omg.CORBA.portable.InputStream is)
       {...}
    public void write_value(
                 org.omg.CORBA.portable.OutputStream os,
                 java.io.Serializable value)
       {...}
    public java.lang.String get_id() {...}
}
```

1.14.2.1 Primitive Type Example

// IDL

valuetype MyLong long;

```
interface foo {
    void bar_in(in MyLong number);
    void bar_inout(inout MyLong number);
};

// Generated Java

public class MyLong implements
            org.omg.CORBA.portable.ValueBase {
    public int value;
    public MyLong(int initial) {value = initial;}
    private static String[] _ids = {intHelper.id() ;}
    public String[] _truncatable_ids () {return _ids;}
}

final public class MyLongHolder
            implements org.omg.CORBA.portable.Streamable {
    public MyLong value;
    public MyLongHolder() {}
    public MyLongHolder(MyLong initial) {...}
    public void _read(
                    org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream os)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

public class MyLongHelper
        implements org.omg.CORBA.portable.BoxedValueHelper {
    public static void insert(org.omg.CORBA.Any a, MyLong t)
        {...}
    public static MyLong extract(Any a) {...}
    public static org.omg.CORBA.TypeCode type() {...}
    public static String id() {...}
    public static MyLong read(
                    org.omg.CORBA.portable.InputStream is)
        {...}
    public static void write(
                    org.omg.CORBA.portable.OutputStream os,
                    MyLong val)
        {...}
    public java.io.Serializable read_value(
                org.omg.CORBA.portable.InputStream is)
        {...}
    public void write_value(
                org.omg.CORBA.portable.OutputStream os,
                java.io.Serializable value)
        {...}
    public java.lang.String get_id() {...}
```

```
}

public interface fooOperations {
   void bar_in(MyLong number);
   void bar_inout(MyLongHolder number);
}

public interface foo extends
      fooOperations,
      org.omg.CORBA.Object,
      org.omg.CORBA.portable.IDLEntity {
}

abstract public class fooHelper {
   public static void insert(
                  org.omg.CORBA.Any a, foo t)
      {...}
   public static foo extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static foo read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
                  org.omg.CORBA.portable.OutputStream os,
                  foo val)
      {...}
   public static foo narrow(
                  java.lang.Object obj)
      {...}
}

final public class fooHolder
      implements org.omg.CORBA.portable.Streamable {
   public foo value;
   public fooHolder() {}
   public fooHolder(foo initial) {...}
   public void _read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                  org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
   }
```

1.14.3 Complex Types

If the value box IDL type is more complex and maps to a Java class (e.g., **string**, **wstring**, **enum**, **struct**, **sequence**, **array**, **any**, **interface**), then the value box type is mapped to the Java class that is appropriate for the IDL type. Holder and helper classes are also generated. The details for the mapped class can be found in the Java Language mapping specification and are not repeated here.

1.14.3.1 Complex Type Example

```
// IDL

valuetype MySequence sequence<long>;

interface foo {
    void bar_in(in MySequence seq);
    void bar_inout(inout MySequence seq);
};
```

```
// Generated Java

final public class MySequenceHolder
         implements org.omg.CORBA.portable.Streamable {
   public int[] value;
   public MySequenceHolder() {}
   public MySequenceHolder(int[] initial) {...}
   public void _read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                  org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}

public class MySequenceHelper
      implements org.omg.CORBA.portable.BoxedValueHelper {
   public static void
      insert(org.omg.CORBA.Any a, int[] t) {...}
   public static int[] extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static int[] read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
                  org.omg.CORBA.portable.OutputStream os,
                  int[] val)
      {...}
   public java.io.Serializable read_value(
```

```
                org.omg.CORBA.portable.InputStream is)
        {...}
    public void write_value(
                org.omg.CORBA.portable.OutputStream os,
                java.io.Serializable value)
        {...}
    public java.lang.String get_id() {...}
}

public interface fooOperations {
    void bar_in(int[] seq);
    void bar_inout(MySequenceHolder seq);
}

public interface foo extends fooOperations,
org.omg.CORBA.Object,
            org.omg.CORBA.portable.IDLEntity {
}

abstract public class fooHelper {
    public static void
        insert(org.omg.CORBA.Any a, foo t) {...}
    public static foo extract(Any a) {...}
    public static org.omg.CORBA.TypeCode type() {...}
    public static String id() {...}
    public static foo read(
                org.omg.CORBA.portable.InputStream is)
        {...}
    public static void write(
                org.omg.CORBA.portable.OutputStream os,
                foo val)
        {...}
    public static foo narrow(java.lang.Object obj); {...}
}
final public class fooHolder
        implements org.omg.CORBA.portable.Streamable {
    public foo value;
    public fooHolder() {}
    public fooHolder(foo initial) {...}
    public void _read(
                org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                org.omg.CORBA.portable.OutputStream os)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
    }
```

1.15 Mapping for Exception

IDL exceptions are mapped very similarly to structs. They are mapped to a Java class that provides instance variables for the fields of the exception and constructors.

CORBA system exceptions are unchecked exceptions. They inherit (indirectly) from **java.lang.RuntimeException**.

User defined exceptions are checked exceptions. They inherit (indirectly) from **java.lang.Exception** via **org.omg.CORBA.UserException** which itself extends **IDLEntity**.

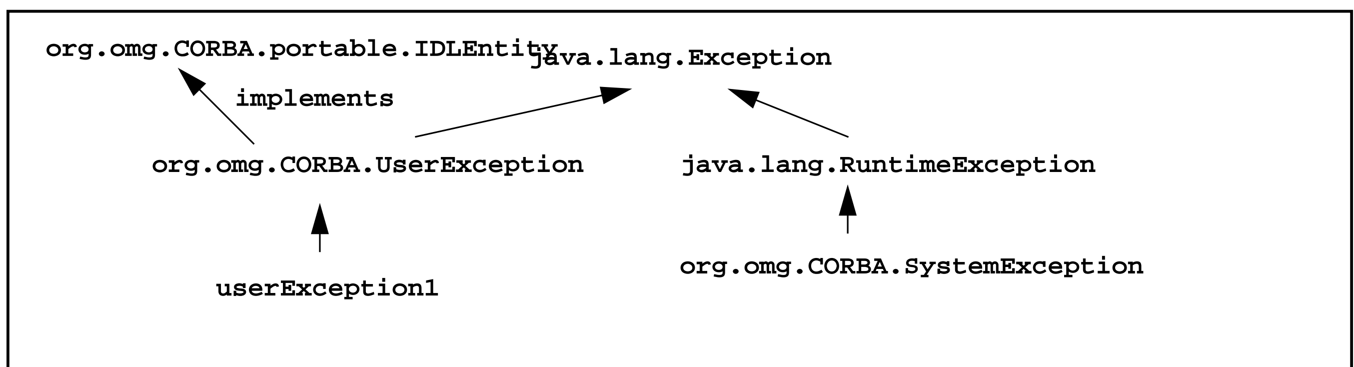

Figure 1-1 Inheritance of Java Exception Classes

1.15.1 User Defined Exceptions

User defined exceptions are mapped to final Java classes that extend **org.omg.CORBA.UserException** and have an "extra full" constructor (described below). They are otherwise mapped just like the IDL **struct** type, including the generation of Helper and Holder classes.

The Java generated exception class has an additional "full" constructor which has an additional initial string reason parameter which is concatenated to the id before calling the base UserException constructor.

If the exception is defined within a nested IDL scope (essentially within an interface), then its Java class name is defined within a special scope. See Section 1.17, "Mapping for Certain Nested Types," on page 1-61 for more details. Otherwise its Java class name is defined within the scope of the Java package that corresponds to the exception's enclosing IDL module.

The definition of the class is as follows:

```
// Java

package org.omg.CORBA;

abstract public class UserException
```

```
        extends java.lang.Exception
        implements org.omg.CORBA.portable.IDLEntity {
    public UserException() {
    super();
    }
public UserException(java.lang.String value) {
     super(value);
    }
}
```

1.15.1.1 Example

```
// IDL

module Example {
    exception ex1 {long reason_code;};
};

// Generated Java

package Example;
final public class ex1 extends org.omg.CORBA.UserException {
    public int reason_code;        // instance
    public ex1() {                 // default constructor
        super(ex1Helper.id());
}
public ex1(int reason_code) { // constructor
    super(ex1Helper.id());
    this.reason_code = reason_code;
}
public ex1(String reason, int reason_code) {// full con-
structor
    super(ex1Helper.id()+" "+reason);
    this.reason_code = reason_code;
}

final public class ex1Holder
        implements org.omg.CORBA.portable.Streamable {
    public ex1 value;
    public ex1Holder() {}
    public ex1Holder(ex1 initial) {...}
    public void _read(
                org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                org.omg.CORBA.portable.OutputStream os)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}
```

1.15.1.2 Unknown User Exception

There is one standard user exception, the unknown user exception. Because the ORB does not know how to create user exceptions, it wraps the user exception as an UnknownUserException and passes it out to the DII layer. The exception is specified as follows:

```
package org.omg.CORBA;
final public class UnknownUserException extends
org.omg.CORBA.UserException {
   public Any except;
   public UnknownUserException() {
      super();
      }
   public UnknownUserException(Any a) {
      super();
      except = a;
   }
}

final public class UnknownUserExceptionHolder
      implements org.omg.CORBA.portable.Streamable {
   public UnknownUserException value;
   public UnknownUserExceptionHolder() {}
   public UnknownUserExceptionHolder(
                  UnknownUserException initial)
      {...}
   public void _read(
                  org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
                  org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}
```

In addition, several exceptions which are PIDL, are also mapped into user exceptions. See Section 1.19.2, “Certain Exceptions,” on page 1-64 for more details.

1.15.2 System Exceptions

The standard IDL system exceptions are mapped to final Java classes that extend **org.omg.CORBA.SystemException** and provide access to the IDL major and minor exception code, as well as a string describing the reason for the exception. Note there are no public constructors for **org.omg.CORBA.SystemException**; only classes that extend it can be instantiated.

The Java class name for each standard IDL exception is the same as its IDL name and is declared to be in the **org.omg.CORBA** package. The default constructor supplies 0 for the minor code, COMPLETED_NO for the completion code, and ““ for the reason

string. There is also a constructor that takes the reason and uses defaults for the other fields, as well as one which requires all three parameters to be specified. The mapping from IDL name to Java class name is listed in the table below.:

Table 1-2 Mapping of IDL Standard Exceptions

IDL Exception	Java Class Name
CORBA::UNKNOWN	`org.omg.CORBA.UNKNOWN`
CORBA::BAD_PARAM	`org.omg.CORBA.BAD_PARAM`
CORBA::NO_MEMORY	`org.omg.CORBA.NO_MEMORY`
CORBA::IMP_LIMIT	`org.omg.CORBA.IMP_LIMIT`
CORBA::COMM_FAILURE	`org.omg.CORBA.COMM_FAILURE`
CORBA::INV_OBJREF	`org.omg.CORBA.INV_OBJREF`
CORBA::NO_PERMISSION	`org.omg.CORBA.NO_PERMISSION`
CORBA::INTERNAL	`org.omg.CORBA.INTERNAL`
CORBA::MARSHAL	`org.omg.CORBA.MARSHAL`
CORBA::INITIALIZE	`org.omg.CORBA.INITIALIZE`
CORBA::NO_IMPLEMENT	`org.omg.CORBA.NO_IMPLEMENT`
CORBA::BAD_TYPECODE	`org.omg.CORBA.BAD_TYPECODE`
CORBA::BAD_OPERATION	`org.omg.CORBA.BAD_OPERATION`
CORBA::NO_RESOURCES	`org.omg.CORBA.NO_RESOURCES`
CORBA::NO_RESPONSE	`org.omg.CORBA.NO_RESPONSE`
CORBA::PERSIST_STORE	`org.omg.CORBA.PERSIST_STORE`
CORBA::BAD_INV_ORDER	`org.omg.CORBA.BAD_INV_ORDER`
CORBA::TRANSIENT	`org.omg.CORBA.TRANSIENT`
CORBA::FREE_MEM	`org.omg.CORBA.FREE_MEM`
CORBA::INV_IDENT	`org.omg.CORBA.INV_IDENT`
CORBA::INV_FLAG	`org.omg.CORBA.INV_FLAG`
CORBA::INTF_REPOS	`org.omg.CORBA.INTF_REPOS`
CORBA::BAD_CONTEXT	`org.omg.CORBA.BAD_CONTEXT`
CORBA::OBJ_ADAPTER	`org.omg.CORBA.OBJ_ADAPTER`
CORBA::DATA_CONVERSION	`org.omg.CORBA.DATA_CONVERSION`
CORBA::OBJECT_NOT_EXIST	`org.omg.CORBA.OBJECT_NOT_EXIST`
CORBA::TRANSACTION_REQUIRED	`org.omg.CORBA.TRANSACTION_REQUIRED`
CORBA::TRANSACTION_ROLLEDBACK	`org.omg.CORBA.TRANSACTION_ROLLEDBACK`
CORBA::INVALID_TRANSACTION	`org.omg.CORBA.INVALID_TRANSACTION`
CORBA::INV_POLICY	`org.omg.CORBA.INV_POLICY`
CORBA::CODESET_INCOMPATIBLE	`org.omg.CORBA.CODESET_INCOMPATIBLE`

The definitions of the relevant classes are specified below.

```
// from org.omg.CORBA package

package org.omg.CORBA;

public final class CompletionStatus {
   // Completion Status constants
   public static final int _COMPLETED_YES = 0,
                           _COMPLETED_NO = 1,
                           _COMPLETED_MAYBE = 2;
   public static final CompletionStatus COMPLETED_YES =
      new CompletionStatus(_COMPLETED_YES);
   public static final CompletionStatus COMPLETED_NO =
      new CompletionStatus(_COMPLETED_NO);
   public static final CompletionStatus COMPLETED_MAYBE =
      new CompletionStatus(_COMPLETED_MAYBE);
   public int value() {...}
   public static final CompletionStatus from_int(int i)
      {...}
   private CompletionStatus(int _value) {...}
}

abstract public class
   SystemException extends java.lang.RuntimeException {
      public int minor;
      public CompletionStatus completed;
      // constructor
      protected SystemException(String reason,
                     int minor,
                     CompletionStatus completed) {
         super(reason);
         this.minor = minor;
         this.completed = completed;
      }
   }

final public class
   UNKNOWN extends org.omg.CORBA.SystemException {
   public UNKNOWN() ...
   public UNKNOWN(int minor, CompletionStatus completed) ...
   public UNKNOWN(String reason) ...
   public UNKNOWN(String reason, int minor,
               CompletionStatus completed)...
}

...

// there is a similar definition for each of the standard
// IDL system exceptions listed in the table above
```

1.15.3 Indirection Exception

The Indirection exception is a Java specific system exception. It is thrown when the ORB's input stream is called to demarshal a value that is encoded as an indirection that is in the process of being demarshaled. This can occur when the ORB input stream calls the **ValueHandler** to demarshal an RMI value whose state contains a recursive reference to itself. Because the top-level **ValueHandler.read_value()** call has not yet returned a value, the ORB input stream's indirection table does not contain an entry for an object with the stream offset specified by the indirection tag. The stream offset is returned in the exception's **offset** field.

The exception is defined as follows:

```
// Java

package org.omg.CORBA.portable;

public class IndirectionException extends org.omg.CORBA.Sys-
temException {
   public int offset;
   public IndirectionException(int offset) {
      super("",
            0,
            org.omg.CORBA.Completion
              Status.COMPLETED_MAYBE);
      this.offset = offset;
   }
}
```

1.16 Mapping for the Any Type

The IDL type **Any** maps to the Java class **org.omg.CORBA.Any** which extends **IDLEntity**. This class has all the necessary methods to insert and extract instances of predefined types. If the extraction operations have a mismatched type, the CORBA::BAD_OPERATION exception is raised.

The **Any** class has an associated helper class. The helper class is in the same Java package as the implementation class for **Any**. Its name is the name of the implementation class concatenated with **Helper**.

Insert and extract methods are defined in order to provide a high speed interface for use by portable stubs and skeletons. An insert and extract method are defined for each primitive IDL type, as well as for a generic streamable to handle the case of non-primitive IDL types. Note that to preserve unsigned type information, unsigned methods are defined where appropriate.

The insert operations set the specified value and reset the any's type if necessary.

The insert and extract methods for **Streamables** implement reference semantics. For the streamable IDL types, an **Any** is a container in which the data is inserted and held. The **Any** does not copy or preserve the state of the streamable object that it holds when

the insert method is invoked. The contents of the **Any** are not serialized until the **write_value()** method is invoked, or the **create_input_stream()** method is invoked. Invoking **create_output_stream()** and writing to the **Any**, or calling **read_value()**, will update the state of the last streamable object that was inserted into the **Any**, if one was previously inserted. Similarly, calling the **extract_streamable()** method multiple times will return the same contained streamable object.

Setting the typecode via the **type()** accessor wipes out the value. An attempt to extract before the value is set will result in a CORBA::BAD_OPERATION exception being raised. This operation is provided primarily so that the type may be set properly for IDL **out** parameters.

```
package org.omg.CORBA;

abstract public class Any implements
                  org.omg.CORBA.portable.IDLEntity {

abstract public boolean equal(org.omg.CORBA.Any a);

// type code accessors
abstract public org.omg.CORBA.TypeCode type();
abstract public void type(org.omg.CORBA.TypeCode t);

// read and write values to/from streams
//     throw exception when typecode inconsistent with value
abstract public void read_value(
   org.omg.CORBA.portable.InputStream is,
   org.omg.CORBA.TypeCode t) throws org.omg.CORBA.MARSHAL;
abstract public void
   write_value(org.omg.CORBA.portable.OutputStream os);

abstract public org.omg.CORBA.portable.OutputStream
         create_output_stream();
abstract public org.omg.CORBA.portable.InputStream
         create_input_stream();

// insert and extract each primitive type

abstract public short    extract_short()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_short(short s);

abstract public int      extract_long()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_long(int i);

abstract public long     extract_longlong()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_longlong(long l);
```

```
abstract public short    extract_ushort()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_ushort(short s);

abstract public int      extract_ulong()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_ulong(int i);

abstract public long     extract_ulonglong()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_ulonglong(long l);

abstract public float    extract_float()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_float(float f);

abstract public double   extract_double()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_double(double d);

abstract public boolean  extract_boolean()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_boolean(boolean b);

abstract public char     extract_char()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_char(char c)
   throws org.omg.CORBA.DATA_CONVERSION;

abstract public char     extract_wchar()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_wchar(char c);

abstract public byte     extract_octet()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_octet(byte b);

abstract public org.omg.CORBA.Any extract_any()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_any(org.omg.CORBA.Any a);

abstract public org.omg.CORBA.Object extract_Object()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void    insert_Object(
                           org.omg.CORBA.Object obj);

abstract public java.io.Serializableextract_Value()
   throws org.omg.CORBA.BAD_OPERATION;
abstract public void    insert_Value(
                                 java.io.Serializable v);
abstract public void    insert_Value(
```

```
                                        java.io.Serializable v,
                                        org.omg.CORBA.TypeCode t)
    throws org.omg.CORBA.MARSHAL;
//      throw exception when typecode inconsistent with value
abstract public void    insert_Object(
                                    org.omg.CORBA.Object obj,
                                    org.omg.CORBA.TypeCode t)
                                        throws
org.omg.CORBA.BAD_PARAM;

abstract public String   extract_string()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_string(String s)
    throwsorg.omg.CORBA.DATA_CONVERSION,
          org.omg.CORBA.MARSHAL;

abstract public String   extract_wstring()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public void     insert_wstring(String s)
    throws org.omg.CORBA.MARSHAL;

// insert and extract typecode

abstract public org.omg.CORBA.TypeCode extract_TypeCode()
    throws org.omg.CORBA.BAD_OPERATION;
abstract public voidinsert_TypeCode(
                                    org.omg.CORBA.TypeCode t);

// Deprecated - insert and extract Principal

/**
*@ deprecated
*/
public org.omg.CORBA.Principal extract_Principal()
        throws org.omg.CORBA.BAD_OPERATION {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
/**
*@ deprecated
*/
public void insert_Principal(org.omg.CORBA.Principal p) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}

// insert and extract non-primitive IDL types
// BAD_INV_ORDER if any doesn't hold a streamable

public org.omg.CORBA.portable.Streamable
        extract_Streamable()
    throws org.omg.CORBA.BAD_INV_ORDER {
```

```
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
public void insert_Streamable(
                  org.omg.CORBA.portable.Streamable s) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}

    // insert and extract fixed

public java.math.BigDecimal extract_fixed() {
    throw org.omg.CORBA.NO_IMPLEMENT();
}

public void insert_fixed(java.math.BigDecimal value) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
public void insert_fixed(
            java.math.BigDecimal value,
            org.omg.CORBA.TypeCode type)
         throws org.omg.CORBA.BAD_INV_ORDER {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

Attempting to insert a **native** type into an **Any** using the **insert_Streamable** method results in a CORBA::MARSHAL exception being raised.

Note – This requires an extra test on every insert (in order to check for native). Say: A CORBA::MARSHAL exception is raised if an attempt is made to marshal an any that contains a native type. (A conforming implementation may choose to raise the exception "early" when the native is first inserted.)

1.17 Mapping for Certain Nested Types

IDL allows type declarations nested within interfaces. Java does not allow classes to be nested within interfaces. Hence those IDL types that map to Java classes and that are declared within the scope of an interface must appear in a special "scope" package when mapped to Java.

IDL interfaces that contain these type declarations will generate a scope package to contain the mapped Java class declarations. The scope package name is constructed by appending **Package** to the IDL type name.

1.17.1 Example

```
// IDL

module Example {
```

```
    interface Foo {
        exception e1 {};
    };
};

// generated Java

package Example.FooPackage;
final public class e1 extends org.omg.CORBA.UserException
     {...}
```

1.18 Mapping for Typedef

Java does not have a typedef construct. Helper classes are generated for all typedefs.

1.18.1 Simple IDL types

IDL types that are mapped to simple Java types may not be subclassed in Java. Hence any typedefs that are type declarations for simple types are mapped to the original (mapped type) everywhere the typedef type appears.

The IDL types covered by this rule are described in Section 1.4, "Mapping for Basic Types," on page 1-6.

1.18.2 Complex IDL types

Typedefs for types that are neither arrays nor sequences are "unwound" to their original type until a simple IDL type or user-defined IDL type (of the non typedef variety) is encountered.

Holder classes are generated for sequence and array typedefs only. The "unwound" type's Holder class is used for the other cases.

1.18.2.1 Example

```
// IDL

struct EmpName {
    string firstName;
    string lastName;
};
typedef EmpName EmpRec;

typedef sequence <long> IntSeq;


// generated Java
//    regular struct mapping for EmpName
//    regular helper class mapping for EmpRec
```

```
//    unwind the sequence

final public class EmpName extends
            org.omg.CORBA.portable.IDLEntity {
   ...
}

public class EmpRecHelper {
   ...
}

final public class IntSeqHolder implements
         org.omg.portable.Streamable {
   ...
}
```

1.19 Mapping Pseudo Objects to Java

1.19.1 Introduction

Pseudo objects are constructs whose definition is usually specified in "IDL," but whose mapping is language specific. A pseudo object is not (usually) a regular CORBA object. Often it is exposed to either clients and/or servers as a process, or a thread local, programming language construct.

For each of the standard IDL pseudo-objects we either specify a specific Java language construct or we specify it as a **pseudo interface**.

This mapping of the pseudo objects was modeled after that in the revised version 1.1 C++ mapping.

1.19.1.1 Pseudo Interface

The use of **pseudo interface** is a convenient device which means that most of the standard language mapping rules defined in this specification may be mechanically used to generate the Java. However, in general the resulting construct is not a CORBA object. Specifically it is:

- not represented in the Interface Repository
- no helper classes are generated
- no holder classes are generated
- mapped to a Java `public abstract class` that does not extend or inherit from any other classes or interfaces

Note – The specific definition given for each piece of PIDL may override the general guidelines above. In such a case, the specific definition takes precedence.

All of the pseudo interfaces are mapped as if they were declared in:

```
module org {
    module omg {
        module CORBA {
            ...
```

That is, they are mapped to the `org.omg.CORBA` Java package.

1.19.2 Certain Exceptions

The standard CORBA PIDL uses several exceptions, **Bounds**, **BadKind**, and **InvalidName**.

No holder and helper classes are defined for these exceptions, nor are they in the interface repository. However, so that users can treat them as "normal exceptions" for programming purposes, they are mapped as normal user exceptions.

They are defined within the scopes that they are used. A **Bounds** and **BadKind** exception are defined in the `TypeCodePackage` for use by **TypeCode**. A **Bounds** exception is defined in the standard CORBA module for use by **NVList**, **ExceptionList**, and **ContextList**. An InvalidName exception is defined in the `ORBPackage` for use by **ORB**.

```
// Java

package org.omg.CORBA;

final public class Bounds
   extends org.omg.CORBA.UserException {
   public Bounds() {...}
}

package org.omg.CORBA.TypeCodePackage;

final public class Bounds
   extends org.omg.CORBA.UserException {
   public Bounds() {...}
}
final public class BadKind
   extends org.omg.CORBA.UserException {
   public BadKind() {...}

}

package org.omg.CORBA.ORBPackage;

final public class InvalidName
   extends org.omg.CORBA.UserException {
   public InvalidName() {...}
}
```

1.19.3 Environment

The **Environment** is used in request operations to make exception information available.

```
// Java code

package org.omg.CORBA;

public abstract class Environment {
   public abstract void exception(
         java.lang.Exception except);
   public abstract java.lang.Exception exception();
   public abstract void clear();
}
```

1.19.4 NamedValue

A **NamedValue** describes a name, value pair. It is used in the DII to describe arguments and return values, and in the context routines to pass property, value pairs.

In Java it includes a name, a value (as an any), and an integer representing a set of flags.

```
typedef unsigned long Flags;
typedef string Identifier;
const Flags ARG_IN = 1;
const Flags ARG_OUT = 2;
const Flags ARG_INOUT = 3;
const Flags CTX_RESTRICT_SCOPE = 15;

pseudo interface NamedValue {
   readonly attribute Identifier name;
   readonly attribute any value;
   readonly attribute Flags flags;
};
```

```
// Java

package org.omg.CORBA;

public interface ARG_IN {
   public static final int value = 1;
}
public interface ARG_OUT {
   public static final int value = 2;
}
public interface ARG_INOUT {
   public static final int value = 3;
}
```

```
public interface CTX_RESTRICT_SCOPE {
   public static final int value = 15;
}

public abstract class NamedValue {
   public abstract String name();
   public abstract Any value();
   public abstract int flags();
}
```

1.19.5 NVList

An **NVList** is used in the DII to describe arguments, and in the context routines to describe context values.

In Java it maintains a modifiable list of **NamedValue**s.

```
pseudo interface NVList {
   readonly attribute unsigned long count;
   NamedValue add(in Flags flags);
   NamedValue add_item(in Identifier item_name, in Flags flags);
   NamedValue add_value(in Identifier item_name,
                        in any val,
                        in Flags flags);
   NamedValue item(in unsigned long index) raises (CORBA::Bounds);
   void remove(in unsigned long index) raises (CORBA::Bounds);
};
```

```
// Java

package org.omg.CORBA;

public abstract class NVList {
   public abstract int count();
   public abstract NamedValue add(int flags);
   public abstract NamedValue add_item(
                     String item_name,
                     int flags);
   public abstract NamedValue add_value(
                     String item_name,
                     Any val,
                     int flags);
   public abstract NamedValue item(int index)
            throws org.omg.CORBA.Bounds;
   public abstract void remove(int index) throws
         org.omg.CORBA.Bounds;
}
```

1.19.6 ExceptionList

An **ExceptionList** is used in the DII to describe the exceptions that can be raised by IDL operations.

It maintains a list of modifiable list of **TypeCode**s.

```
pseudo interface ExceptionList {
    readonly attribute unsigned long count;
    void add(in TypeCode exc);
    TypeCode item (in unsigned long index) raises (CORBA::Bounds);
    void remove (in unsigned long index) raises (CORBA::Bounds);
};
```

```
// Java

package org.omg.CORBA;

public abstract class ExceptionList {
    public abstract int count();
    public abstract void add(TypeCode exc);
    public abstract TypeCode item(int index)
             throws org.omg.CORBA.Bounds;
    public abstract void remove(int index)
             throws org.omg.CORBA.Bounds;
}
```

1.19.7 Context

A **Context** is used in the DII to specify a context in which context strings must be resolved before being sent along with the request invocation.

```
pseudo interface Context {
    readonly attribute Identifier context_name;
    readonly attribute Context parent;
    Context create_child(in Identifier child_ctx_name);
    void set_one_value(in Identifier propname, in any propvalue);
    void set_values(in NVList values);
    void delete_values(in Identifier propname);
    NVList get_values(in Identifier start_scope,
                      in Flags op_flags,
                      in Identifier pattern);
};

pseudo interface ContextList {
    readonly attribute unsigned long count;
    void add(in string ctx);
    string item(in unsigned long index) raises (CORBA::Bounds);
    void remove(in unsigned long index) raises (CORBA::Bounds);
};
```

```
// Java

package org.omg.CORBA;

public abstract class Context {
   public abstract String context_name();
   public abstract Context parent();
   public abstract Context create_child(
                              String child_ctx_name);
   public abstract void set_one_value(
                              String propname,
                              Any propvalue);
   public abstract void set_values(
                              NVList values);
   public abstract void delete_values(
                              String propname);
   public abstract NVList get_values(
                              String start_scope,
                              int op_flags,
                              String pattern);
}

public abstract class ContextList {
   public abstract int count();
   public abstract void add(String ctx);
   public abstract String item(int index)
      throws org.omg.CORBA.Bounds;
   public abstract void remove(int index)
      throws org.omg.CORBA.Bounds;
}
```

1.19.8 Request

A **Request** is used in the DII to describe an invocation.

```
pseudo interface Request {
   readonly attribute Object target;
   readonly attribute Identifier operation;
   readonly attribute NVList arguments;
   readonly attribute NamedValue result;
   readonly attribute Environment env;
   readonly attribute ExceptionList exceptions;
   readonly attribute ContextList contexts;

   attribute Context ctx;

   any add_in_arg();
```

```
    any add_named_in_arg(in string name);
    any add_inout_arg();
    any add_named_inout_arg(in string name);
    any add_out_arg();
    any add_named_out_arg(in string name);
    void set_return_type(in TypeCode tc);
    any return_value();

    void invoke();
    void send_oneway();
    void send_deferred();
    void get_response();
    boolean poll_response();
};

// Java

package org.omg.CORBA;

public abstract class Request {

    public abstract Object target();
    public abstract String operation();
    public abstract NVList arguments();
    public abstract NamedValue result();
    public abstract Environment env();
    public abstract ExceptionList exceptions();
    public abstract ContextList contexts();

    public abstract Context ctx();
    public abstract void ctx(Context c);

    public abstract Any add_in_arg();
    public abstract Any add_named_in_arg(String name);
    public abstract Any add_inout_arg();
    public abstract Any add_named_inout_arg(String name);
    public abstract Any add_out_arg();
    public abstract Any add_named_out_arg(String name);
    public abstract void set_return_type(TypeCode tc);
    public abstract Any return_value();

    public abstract void invoke();
    public abstract void send_oneway();
    public abstract void send_deferred();
    public abstract void get_response()
          throws org.omg.CORBA.WrongTransaction;
    public abstract boolean poll_response();
}
```

It is permissible to call the `return_value()` method before issuing the **Request**.(e.g., before calling `invoke()`, `send_oneway()`, or `send_deferred()`. Changes made to the **Any** that stores the result may be used by the implementation to improve performance. For example, one may insert a `Streamable` into the **Any** containing the return value before invoking the **Request**. Because **Any**s provide reference semantics, the result will be marshaled directly into the `Streamable` object avoiding additional marshalling if the **Any** were extracted after invocation.

1.19.9 TypeCode

The deprecated **parameter** and **param_count** methods are not mapped. The Typecode has a holder and a helper class.

The helper class is in the same Java package as the implementation class for TypeCode. Its name is the name of the implementation class concatenated with `Helper`.

```
module CORBA {

enum TCKind {
    tk_null, tk_void,
    tk_short, tk_long, tk_ushort, tk_ulong,
    tk_float, tk_double, tk_boolean, tk_char,
    tk_octet, tk_any, tk_TypeCode, tk_Principal, tk_objref,
    tk_struct, tk_union, tk_enum, tk_string,
    tk_sequence, tk_array, tk_alias, tk_except,
    tk_longlong, tk_ulonglong, tk_longdouble,
    tk_wchar, tk_wstring, tk_fixed,
    tk_value, tk_value_box,
    tk_native,
    tk_abstract_interface
};

typedef short ValueModifier;
const ValueModifier VM_NONE = 0;
const ValueModifier VM_CUSTOM = 1;
const ValueModifier VM_ABSTRACT = 2;
const ValueModifier VM_TRUNCATABLE = 3;

typedef short Visibility;
const Visibility PRIVATE_MEMBER = 0;
const Visibility PUBLIC_MEMBER = 1;

};
```

```
// Java

package org.omg.CORBA;
```

```
public class TCKind {
   public static final int _tk_null = 0;
   public static final
      TCKind tk_null = new TCKind(_tk_null);
   public static final int _tk_void = 1;
      TCKind tk_void = new TCKind(_tk_void);
   public static final int _tk_short = 2;
      TCKind tk_short = new TCKind(_tk_short);
   public static final int _tk_long = 3;
      TCKind tk_long = new TCKind(_tk_long);
   public static final int _tk_ushort = 4;
      TCKind tk_ushort = new TCKind(_tk_ushort);
   public static final int _tk_ulong = 5;
      TCKind tk_ulong = new TCKind(_tk_ulong);
   public static final int _tk_float = 6;
      TCKind tk_float = new TCKind(_tk_float);
   public static final int _tk_double = 7;
      TCKind tk_double = new TCKind(_tk_double);
   public static final int _tk_boolean = 8;
      TCKind tk_boolean = new TCKind(_tk_boolean);
   public static final int _tk_char = 9;
      TCKind tk_char = new TCKind(_tk_char);
   public static final int _tk_octet = 10;
      TCKind tk_octet = new TCKind(_tk_octet);
   public static final int _tk_any = 11;
      TCKind tk_any = new TCKind(_tk_any);
   public static final int _tk_TypeCode = 12;
      TCKind tk_TypeCode = new TCKind(_tk_TypeCode);
   public static final int _tk_Principal = 13;
      TCKind tk_Principal = new TCKind(_tk_Principal);
   public static final int _tk_objref = 14;
      TCKind tk_objref = new TCKind(_tk_objref);
   public static final int _tk_stuct = 15;
      TCKind tk_stuct = new TCKind(_tk_stuct);
   public static final int _tk_union = 16;
      TCKind tk_union = new TCKind(_tk_union);
   public static final int _tk_enum = 17;
      TCKind tk_enum = new TCKind(_tk_enum);
   public static final int _tk_string = 18;
      TCKind tk_string = new TCKind(_tk_string);
   public static final int _tk_sequence = 19;
      TCKind tk_sequence = new TCKind(_tk_sequence);
   public static final int _tk_array = 20;
      TCKind tk_array = new TCKind(_tk_array);
   public static final int _tk_alias = 21;
      TCKind tk_alias = new TCKind(_tk_alias);
   public static final int _tk_except = 22;
      TCKind tk_except = new TCKind(_tk_except);
   public static final int _tk_longlong = 23;
      TCKind tk_longlong = new TCKind(_tk_longlong);
   public static final int _tk_ulonglong = 24;
```

```
        TCKind tk_ulonglong = new TCKind(_tk_ulonglong);
    public static final int _tk_longdouble = 25;
        TCKind tk_longdouble = new TCKind(_tk_longdouble);
    public static final int _tk_wchar = 26;
        TCKind tk_wchar = new TCKind(_tk_wchar);
    public static final int _tk_wstring = 27;
        TCKind tk_wstring = new TCKind(_tk_wstring);
    public static final int _tk_fixed = 28;
        TCKind tk_fixed = new TCKind(_tk_fixed);
    public static final int _tk_value = 29;
        TCKind tk_value = new TCKind(_tk_value);
    public static final int _tk_value_box = 30;
        TCKind tk_value_box = new TCKind(_tk_valuebox);
    public static final int _tk_native = 31;
        TCKind tk_native= new TCKind(_tk_native);
    public static final int _tk_abstract_interface = 32;
        TCKind tk_abstract_interface = new TCK-
ind(_tk_abstract_interface);

    public int value() {...}
    public static TCKind from_int(int value) {...}
    protected TCKind(int value) {...}
}

public interface VM_NONE {
    short value = 0;
}
public interface VM_CUSTOM {
    short value = 1;
}
public interface VM_ABSTRACT {
    short value = 2;
}
public interface VM_TRUNCATABLE {
    short value = 3;
}
public interface PRIVATE_MEMBER {
    short value = 0;
}
public interface PUBLIC_MEMBER {
    short value = 1;
}
```

pseudo interface TypeCode {

exception Bounds {};
exception BadKind {};

// for all TypeCode kinds

```
boolean equal(in TypeCode tc);

boolean equivalent(in TypeCode tc);
TypeCode get_compact_typecode();


TCKind kind();

// for objref, struct, union, enum, alias, value, valuebox,
//    native, abstract_interface, and except
RepositoryID id() raises (BadKind);
Identifier name() raises (BadKind);

// for struct, union, enum, value, and except
unsigned long member_count() raises (BadKind);
Identifier member_name(in unsigned long index)
      raises (BadKind, Bounds);

// for struct, union, value, and except
TypeCode member_type(in unsigned long index)
     raises (BadKind, Bounds);

// for union
any member_label(in unsigned long index) raises (BadKind, Bounds);
TypeCode discriminator_type() raises (BadKind);
long default_index() raises (BadKind);

// for string, sequence, and array
unsigned long length() raises (BadKind);

// for sequence, array, value, value_box and alias
TypeCode content_type() raises (BadKind);

// for fixed
unsigned short fixed_digits() raises(BadKind);
short fixed_Scale() raised (BadKind);

// for value
Visibility member_visibility(in unsigned long index)
     raises (BadKind, Bounds);
ValueModifier type_modifier() raises (BadKind);
TypeCode concrete_base_type() raises (BadKind);

}
```

```
// Java

package org.omg.CORBA;

final public class TypeCodeHolder
      implements org.omg.CORBA.portable.Streamable {
```

```
    public Typecode value;
    public TypeCodeHolder() {}
    public TypeCodeHolder(Typecode initial) {...}
    public void _read(
                    org.omg.CORBA.portable.InputStream is)
        {...}
    public void _write(
                    org.omg.CORBA.portable.OutputStream os)
        {...}
    public org.omg.CORBA.TypeCode _type() {...}
}

public abstract class TypeCode extends
            org.omg.CORBA.portable.IDLEntity {

    // for all TypeCode kinds
    public abstract boolean equal(TypeCode tc);
    public abstract boolean equivalent(TypeCode tc);
    public abstract TypeCode get_compact_typecode();
    public abstract TCKind kind();

    // for objref, struct, union, enum, alias,
    //    value, value_box, native,
    //    abstract_interface, and except
    public abstract String id() throws
                TypeCodePackage.BadKind;
    public abstract String name() throws
                TypeCodePackage.BadKind;

    // for struct, union, enum, value, and except
    public abstract int member_count() throws
                TypeCodePackage.BadKind;
    public abstract String member_name(int index) throws
          TypeCodePackage.BadKind, TypeCodePackage.Bounds;

    // for struct, union, value, and except
    public abstract TypeCode member_type(int index)
          throws TypeCodePackage.BadKind,
                   TypeCodePackage.Bounds;

    // for union
    public abstract Any member_label(int index)
          throws TypeCodePackage.BadKind,
                   TypeCodePackage.Bounds;
    public abstract TypeCode discriminator_type()
          throws TypeCodePackage.BadKind;
    public abstract int default_index()
          throws TypeCodePackage.BadKind;

    // for string, sequence, and array
    public abstract int length()
```

```
            throws TypeCodePackage.BadKind;

    // for sequence, array, value, value_box and alias
    public abstract TypeCode content_type()
        throws TypeCodePackage.BadKind;

    // for fixed
    public abstract short fixed_digits()
        throws TypeCodePackage.BadKind;
    public abstract short fixed_Scale()
        throws TypeCodePackage.BadKind;

    // for value
    public abstract short member_visibility(long index)
        throws   TypeCodePackage.BadKind,
                 TypeCodePackage.Bounds;
    public abstract short type_modifer()
        throws TypeCodePackage.BadKind;
    public abstract TypeCode concrete_base_type()
        throws TypeCodePackage.BadKind;
}
```

1.19.10 ORB

The **ORB** defines operations that are implemented by the ORB core and are in general not dependent upon a particular object or object adapter.

In addition to the operations specifically defined on the ORB in the core, additional methods needed specifically for Java are also defined.

The **UnionMemberSeq**, **EnumMemberSeq**, **StructMemberSeq, ValueMemberSeq** typedefs are real IDL and bring in the Interface Repository (see *The Common Object Request Broker: Architecture and Specification*, *Interface Repository* chapter). The **ServiceInformation** struct is real IDL and is defined in *The Common Object Request Broker: Architecture and Specification*, *ORB Interface* chapter. Rather than tediously listing interfaces, and other assorted types, suffice it to say that these constructs are all mapped following the rules for IDL set forth in this specification.

```
pseudo interface ORB {                                  // PIDL
    typedef string ObjectId;
    typedef sequence <ObjectId> ObjectIdList;

    exception InconsistentTypeCode {};

    exception InvalidName {};

    string object_to_string (
```

```
    in Object           obj
);

Object string_to_object (
    in string           str
);

// Dynamic Invocation related operations

void create_list (
    in long             count,
    out NVList          new_list
);

void create_operation_list (
    in OperationDef     oper,
    out NVList          new_list
);

void get_default_context (
    out Context         ctx
);

void send_multiple_requests_oneway(in RequestSeq req);

void send_multiple_requests_deferred(in RequestSeq req);

boolean poll_next_response();

void get_next_response(out Request req);

// Service information operations

boolean get_service_information (
    in ServiceType service_type,
    out ServiceInformation service_information
);

ObjectIdList list_initial_services ();

// Initial reference operation

Object resolve_initial_references (
    in ObjectId identifier
) raises (InvalidName);

// Type code creation operations

TypeCode create_struct_tc (
```

```
    in RepositoryId id,
    in Identifier name,
    in StructMemberSeq members
);

TypeCode create_union_tc (
    in RepositoryId id,
    in Identifier name,
    in TypeCode discriminator_type,
    in UnionMemberSeq members
);

TypeCode create_enum_tc (
    in RepositoryId id,
    in Identifier name,
    in EnumMemberSeq members
);

TypeCode create_alias_tc (
    in RepositoryId id,
    in Identifier name,
    in TypeCode original_type
);

TypeCode create_exception_tc (
    in RepositoryId id,
    in Identifier name,
    in StructMemberSeq members
);

TypeCode create_interface_tc (
    in RepositoryId id,
    in Identifier name
);

TypeCode create_string_tc (
    in unsigned long bound
);

TypeCode create_wstring_tc (
    in unsigned long bound
);

TypeCode create_fixed_tc (
    in unsigned short digits,
    in short scale
);

TypeCode create_sequence_tc (
```

```
    in unsigned long bound,
    in TypeCode element type
);

TypeCode create_recursive_sequence_tc ( // deprecated
    in unsigned long bound,
    in unsigned long offset
);

TypeCode create_array_tc (
    in unsigned long length,
    in TypeCode element_type
);

TypeCode create_value_tc (
    in RepositoryId         id,
    in Identifier           name,
    in ValueModifier        type_modifier,
    in TypeCode             concrete_base,
    in ValueMemberSeq       members
);

TypeCode create_value_box_tc (
    in RepositoryId         id,
    in Identifier           name,
    in TypeCode             boxed_type
);

TypeCode create_native_tc (
    in RepositoryId         id,
    in Identifier           name
);

TypeCode create_recursive_tc (
    in RepositoryId         id
);

TypeCode create_abstract_interface_tc (
    in RepositoryId         id,
    in Identifier           name
);


// Thread related operations

boolean work_pending();

void perform_work();

void run();
```

```
        void shutdown(
            in boolean          wait_for_completion
        );

        void destroy();

        // Policy related operations

        Policy create_policy(
            in PolicyType       type,
            in any              val
        ) raises (PolicyError);

        // Dynamic Any related operations deprecated and removed
        // from primary list of ORB operations

        // Value factory operations

        ValueFactory register_value_factory(
                        in RepositoryId id,
                        in ValueFactory factory
        );
        void unregister_value_factory(in RepositoryId id);
        ValueFactory lookup_value_factory(in RepositoryId id);

// Additional operations that only appear in the Java mapping

    TypeCode get_primitive_tc(in TCKind tcKind);
    ExceptionList create_exception_list();
    ContextList create_context_list();
    Environment create_environment();
    Current get_current();
    Any create_any();
    OutputStream create_output_stream();
    void connect(Object obj);
    void disconnect(Object obj);
    Object get_value_def(in String repid);
    void set_delegate(Object wrapper);

// additional methods for ORB initialization go here, but only
```

```
//    appear in the mapped Java (seeSection 1.21.9, “ORB Initialization)
// Java signatures
//    public static ORB init(Strings[] args, Properties props);
//    public static ORB init(Applet app, Properties props);
//    public static ORB init();
// abstract protected void set_parameters(String[] args,
//                                         java.util.Properties props);
// abstract protected void set_parameters(java.applet.Applet app,
//                                         java.util.Properties props);

};
```

All types defined in this chapter are either part of the CORBA or the CORBA_2_3 module. When referenced in OMG IDL, the type names must be prefixed by "**CORBA::**" or "**CORBA_2_3::**".

```
// Java

package org.omg.CORBA;

public abstract class ORB {

   public abstract org.omg.CORBA.Object
         string_to_object(String str);

   public abstract String
         object_to_string(org.omg.CORBA.Object obj);

   // Dynamic Invocation related operations

   public abstract NVList create_list(int count);

   /**
   *@deprecated Deprecated by CORBA 2.3.
   */
   public abstract NVList create_operation_list(
                              OperationDef oper);

   public NVList create_operation_list(
                              org.omg.CORBA.Object oper);
   // oper must really be an OperationDef

   public abstract NamedValue create_named_value(
                              String name,
                              Any value,
                              int flags);

   public abstract ExceptionList create_exception_list();

   public abstract ContextList create_context_list();
```

```
public abstract Context get_default_context();

public abstract Environment create_environment();

public abstract void send_multiple_requests_oneway(
                              Request[] req);

public abstract void send_multiple_requests_deferred(
                              Request[] req);

public abstract boolean poll_next_response();

public abstract Request get_next_response() throws
      org.omg.CORBA.WrongTransaction;

// Service information operations

public boolean get_service_information(
            short service_type,
            ServiceInformationHolder service_info) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

public abstract String[] list_initial_services();

// Initial reference operation

public abstract org.omg.CORBA.Object
   resolve_initial_references(
                           String object_name)
   throws org.omg.CORBA.ORBPackage.InvalidName;

// typecode creation

public abstract TypeCode create_struct_tc(
                           String id,
                           String name,
                           StructMember[] members);

public abstract TypeCode create_union_tc(
                           String id,
                           String name,
                           TypeCode discriminator_type,
                           UnionMember[] members);

public abstract TypeCode create_enum_tc(
                           String id,
                           String name,
                           String[] members);

public abstract TypeCode create_alias_tc(
```

```
                                String id,
                                String name,
                                TypeCode original_type);

public abstract TypeCode create_exception_tc(
                                String id,
                                String name,
                                StructMember[] members);

public abstract TypeCode create_interface_tc(
                                String id,
                                String name);

public abstract TypeCode create_string_tc(int bound);

public abstract TypeCode create_wstring_tc(int bound);

public TypeCode create_fixed_tc(
                                short digits,
                                short scale) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

public abstract TypeCode create_sequence_tc(
                                int bound,
                                TypeCode element_type);

/**
*@deprecated Deprecated by CORBA 2.3.
*/
public abstract TypeCode create_recursive_sequence_tc(
                                int bound,
                                int offset);

public abstract TypeCode create_array_tc(
                                int length,
                                TypeCode element_type);

public TypeCode create_value_tc(
                                String id,
                                String name,
                                short type_modifier,
                                TypeCode concrete_base,
                                ValueMember[] members) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

public TypeCode create_value_box_tc(
                                String id,
                                String name,
                                TypeCode boxed_type) {
```

```
    throw new org.omg.CORBA.NO_IMPLEMENT();
}

public TypeCode create_native_tc(
                            String id,
                            String name) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}

public TypeCode create_recursive_tc(
                            String id) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}

public TypeCode create_abstract_interface_tc(
                            String id,
                            String name) {
    throw org.omg.CORBA.NO_IMPLEMENT();
}

/**
*@deprecated Deprecated by CORBA 2.2.
*/
public Current get_current() {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}

/**
*@deprecated Deprecated by Portable Object Adapter,
see OMG document orbos/98-01-06 for details.
*/
public void connect(        org.omg.CORBA.Object obj) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}

/**
*@deprecated Deprecated by Portable Object Adapter,
see OMG document orbos/98-01-06 for details.
*/
public void disconnect(     org.omg.CORBA.Object obj) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}

// Thread related operations

public boolean work_pending() {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}

public void perform_work() {
    throw new org.omg.CORBA.NO_IMPLEMENT();
```

```
    }
    public void run() {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }

    public void shutdown(boolean wait_for_completion) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public void destroy() {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }

    // Policy related operations

    public Policy create_policy(short policy_type, Any val)
        throws org.omg.CORBA.PolicyError {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }

    // additional methods for IDL/Java mapping

    public abstract TypeCode get_primitive_tc(TCKind tcKind);

    public abstract Any create_any();

    public abstract org.omg.CORBA.portable.OutputStream
        create_output_stream();

    // additional static methods for ORB initialization

    public static ORB init(
                            Strings[] args,
                            Properties props);
    public static ORB init(
                            Applet app,
                            Properties props);
    public static ORB init();
    abstract protected void set_parameters(
                            String[] args,
                            java.util.Properties props);
    abstract protected void set_parameters(
                            java.applet.Applet app,
                            java.util.Properties props);

}

package org.omg.CORBA_2_3;

public abstract class ORB extends org.omg.CORBA.ORB {
```

```
    // always return a ValueDef or throw BAD_PARAM if
    // repid not of a value
    public org.omg.CORBA.Object get_value_def(
                                String repid)
          throws org.omg.CORBA.BAD_PARAM {
       throw new org.omg.CORBA.NO_IMPLEMENT();
    }

    // Value factory operations

    public org.omg.CORBA.portable.ValueFactory
          register_value_factory(
                String id,
                org.omg.CORBA.portable.ValueFactory factory){
       throw new org.omg.CORBA.NO_IMPLEMENT();
    }

    public void unregister_value_factory(String id) {
       throw new org.omg.CORBA.NO_IMPLEMENT();
    }

    public org.omg.CORBA.portable.ValueFactory
       lookup_value_factory(String id) {
          throw new org.omg.CORBA.NO_IMPLEMENT();
    }

     public void set_delegate(java.lang.Object wrapper) {
       throw new org.omg.CORBA.NO_IMPLEMENT();
    }
}
```

1.19.10.1 set_delegate

The **set_delegate()** method supports the Java ORB portability interfaces by providing a method for classes that support ORB portability through delegation to set their delegate. This is typically required in cases where instances of such classes were created by the application programmer rather than the ORB runtime. The wrapper parameter is the instance of the object on which the ORB must set the delegate. The mechanism to set the delegate is specific to the class of the wrapper instance. The **set_delegate()** method supports setting delegates on instances of the following Java classes:

org.omg.PortableServer.Servant

If the wrapper parameter is not an instance of a class for which the ORB can set the delegate, the CORBA::BAD_PARAM exception is thrown.

1.19.10.2 get_value_def

The **get_value_def()** method is declared to return an **org.omg.CORBA.Object**. However, it is intended to only be used for value types. The actual implementation:

- raises the BAD_PARAM system exception if the specified **repid** parameter does not identify an IDL type that is a value type.
- returns a **ValueDef** if the specified **repid** parameter identifies an IDL type that is a value type.

1.19.11 CORBA::Object

The IDL **Object** type is mapped to the **org.omg.CORBA.Object** and **org.omg.CORBA.ObjectHelper** classes as shown below.

The Java interface for each user defined IDL **interface** extends **org.omg.CORBA.Object**, so that any object reference can be passed anywhere a **org.omg.CORBA.Object** is expected.

The **Policy**, **DomainManager**, and **SetOverrideType** types are real IDL and are defined in *The Common Object Request Broker: Architecture and Specification*, *ORB Interface* chapter. Rather than tediously list the mapping here, suffice it to say that these constructs are all mapped following the rules for IDL set forth in this specification.

```
// Java

package org.omg.CORBA;

public interface Object {
   boolean _is_a(String Identifier);
   boolean _is_equivalent(Object that);
   boolean _non_existent();
   int _hash(int maximum);
   org.omg.CORBA.Object _duplicate();
   void _release();
   /**
   *@deprecated Deprecated by CORBA 2.3.
   */
   InterfaceDef _get_interface();
   org.omg.CORBA.Object _get_interface_def();
   Request _request(String s);
   Request _create_request(Context ctx,
                           String operation,
                           NVList arg_list,
                           NamedValue result);
   Request _create_request(Context ctx,
                           String operation,
                           NVList arg_list,
```

```
                                NamedValue result,
                                ExceptionList exclist,
                                ContextList ctxlist);
   Policy _get_policy(int policy_type);
   DomainManager[] _get_domain_managers();
   org.omg.CORBA.Object _set_policy_override(
                                Policy[] policies,
                                SetOverrideType set_add);
}

abstract public class ObjectHelper {
   public static void insert(
                     org.omg.CORBA.Any a,
                     org.omg.CORBA.Object t)
      {...}
   public static org.omg.CORBA.Object extract(Any a) {...}
   public static org.omg.CORBA.TypeCode type() {...}
   public static String id() {...}
   public static org.omg.CORBA.Object read(
                     org.omg.CORBA.portable.InputStream is)
      {...}
   public static void write(
                     org.omg.CORBA.portable.OutputStream os,
                     org.omg.CORBA.Object val)
      {...}
}
```

1.19.12 Principal

Principal was deprecated in CORBA 2.2. The "old" mapping is preserved and documented here for the purpose of maintaining binary compatibility, but all methods and classes associated with **Principal** are documented as "deprecated" in Java, and conforming implementations may raise a NO_IMPLEMENT exception. (A product which implements other behavior is considered to be implementing a proprietary vendor extension.)

```
pseudo interface Principal {
attribute sequence<octet> name;
}
```

```
// Java

package org.omg.CORBA;

/**
* @deprecated Principal
*/
public class Principal {
   /**
   *@deprecated Deprecated by CORBA 2.2.
```

```
    */
    public byte[] name() {
       throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    /**
    *@deprecated Deprecated by CORBA 2.2.
    */
    public void name(byte[] name) {
       throw new org.omg.CORBA.NO_IMPLEMENT();
    }
}
```

1.20 Server-Side Mapping

1.20.1 Introduction

This section discusses how object implementations written in Java create and register objects with the ORB runtime.

This chapter was substantially changed from the previous preliminary specification after the final adoption of the server side ORB Portability IDL/Java Language Mapping. The previous mapping is officially deprecated and is no longer part of CORBA. However, the new mapping was carefully designed so that it is possible for a particular vendor's implementation to support both as a migration aid to users. Some of the "old" classes are still required to be present (and are included in this specification) in order to preserve binary compatibility, but now have default implementations which throw a NO_IMPLEMENT exception.

1.20.2 Implementing Interfaces

To define an implementation in Java, a developer must write an implementation class. Instances of the implementation class implement IDL interfaces. The implementation class must define public methods corresponding to the operations and attributes of the IDL interface supported by the object implementation, as defined by the mapping specification for IDL interfaces. Providing these methods is sufficient to satisfy all abstract methods defined by a particular interface's skeleton class.

The mapping specifies two alternative relationships between the application-supplied implementation class and the generated class or classes for the interface. Specifically, the mapping requires support for both *inheritance-based* relationships and *delegation-based* relationships. Conforming ORB implementations shall provide both of these alternatives. Conforming applications may use either or both of these alternatives.

1.20.2.1 Mapping of PortableServer::Servant

The **PortableServer** module for the Portable Object Adapter (POA) defines the native **Servant** type. In Java, the **Servant** type is mapped to the Java **org.omg.PortableServer.Servant** class. This class is defined as follows:

```
//Java

package org.omg.PortableServer;
import org.omg.CORBA.ORB;
import org.omg.PortableServer.POA;

abstract public class Servant {

   // Convenience methods for application programmer
   final public org.omg.CORBA.Object _this_object() {
      return _get_delegate().this_object(this);
   }

   final public org.omg.CORBA.Object _this_object(ORB orb) {
      try {
         ((org.omg.CORBA_2_3.ORB)orb).set_delegate(this);
      }
      catch(ClassCastException e) {
         throw new org.omg.CORBA.BAD_PARAM(
"POA Servant requires an instance of org.omg.CORBA_2_3.ORB"
                                       );
      }
       return _this_object();
   }

   final public ORB _orb() {
      return _get_delegate().orb(this);
   }

   final public POA _poa() {
      return _get_delegate().poa(this);
   }

   final public byte[] _object_id() {
      return _get_delegate().object_id(this);
   }

   // Methods which may be overridden by the
   //    application programmer

   public POA _default_POA() {
      return _get_delegate().default_POA(this);
   }

   public boolean _is_a(String repository_id) {
      return _get_delegate().is_a(this, repository_id);
   }

   public boolean _non_existent() {
      return _get_delegate().non_existent(this);
   }
```

```
   public org.omg.CORBA.InterfaceDef _get_interface() {
      return _get_delegate().get_interface(this);
   }

   // methods for which the skeleton or application
   //    programmer must provide an an implementation

   abstract public String[] _all_interfaces(
                                    POA poa,
                                    byte[] objectId);

   // private implementation methods

   private transient Delegate _delegate = null;

   final public Delegate _get_delegate() {
      if (_delegate == null) {
         throw new org.omg.CORBA.BAD_INV_ORDER(
"The Servant has not been associated with an ORBinstance");
      }
      return _delegate;
   }

   final public void _set_delegate(Delegate delegate) {
      _delegate = delegate;
   }
}
```

The **Servant** class is a Java abstract class which serves as the base class for all POA servant implementations. It provides a number of methods that may be invoked by the application programmer, as well as methods which are invoked by the POA itself and may be overridden by the user to control aspects of servant behavior.

With the exception of the **_all_interfaces()** and **_this_object(ORB orb)** methods, all methods defined on the Servant class may only be invoked after the Servant has been associated with an ORB instance. Attempting to invoke the methods on a Servant that has not been associated with an ORB instance results in a CORBA::BAD_INV_ORDER exception being raised.

A Servant may be associated with an ORB instance via one of the following means:

- Through a call to **_this_object(ORB orb)** passing an ORB instance as parameter. The Servant will become associated with the specified ORB instance.
- By explicitly activating a Servant with a POA by calling either **POA::activate_object** or **POA::activate_object_with_id**. Activating a Servant in this fashion will associate the Servant with the ORB instance, which contains the POA on which the Servant has been activated.

- By returning a Servant instance from a ServantManager. The Servant returned from **PortableServer::ServantActivator::incarnate()** or **PortableServer::ServantLocator::preinvoke()** will be associated with the ORB instance that contains the POA on which the ServantManager is installed.
- By installing the Servant as a default servant on a POA. The Servant will become associated with the ORB instance which contains the POA for which the Servant is acting as a default servant.
- By explicitly setting it by a call to **`org.omg.CORBA_2_3.ORB.set_delegate()`**.

It is not possible to associate a Servant with more than one ORB instance at a time. Attempting to associate a Servant with more than one ORB instance will result in undefined behavior.

_this_object

The **`_this_object()`** methods have the following purposes:

- Within the context of a request invocation on the target object represented by the servant, it allows the servant to obtain the object reference for the target CORBA Object it is incarnating for that request. This is true even if the servant incarnates multiple CORBA objects. In this context, **`_this_object()`** can be called regardless of the policies the dispatching POA was created with.
- Outside the context of a request invocation on the target object represented by the servant, it allows a servant to be implicitly activated if its POA allows implicit activation. This requires the POA to have been created with the **IMPLICIT_ACTIVATION** policy. If the POA was not created with the **IMPLICIT_ACTIVATION** policy, the CORBA::OBJ_ADAPTER exception is thrown. The POA to be used for implicit activation is determined by invoking the servant's **`_default_POA()`** method.
- Outside the context of a request invocation on the target object represented by the servant, it will return the object reference for a servant that has already been activated, as long as the servant is not incarnating multiple CORBA objects. This requires the servant's POA to have been created with the **UNIQUE_ID** and **RETAIN** policies. If the POA was created with the **MULITPLE_ID** or **NON_RETAIN** policies, the CORBA::OBJ_ADAPTER exception is thrown. The POA used in this operation is determined by invoking the servant's **`_default_POA()`** method.
- The **`_this_object(ORB orb)`** method first associates the Servant with the specified ORB instance and then invokes **`_this_object()`** as normal.

_orb

The **`_orb()`** method is a convenience method that returns the instance of the ORB currently associated with the Servant.

_poa and _object_id

The methods **`_poa()`** and **`_object_id()`** are equivalent to calling the methods **PortableServer::Current:get_POA** and **PortableServer::Current::get_object_id**. If the **PortableServer::Current** object throws a PortableServer::Current::NoContext exception, then **`_poa()`** and **`_object_id()`** throws a CORBA::OBJ_ADAPTER system exception instead. These methods are provided as a convenience to the user to allow easy execution of these common methods.

_default_POA

The method **`_default_POA()`** returns a default POA to be used for the servant outside the context of POA invocations. The default behavior of this function is to return the root POA from the ORB instance associated with the servant. Subclasses may override this method to return a different POA. It is illegal to return a null value.

_all_interfaces

The **`_all_interfaces()`** method is used by the ORB to obtain complete type information from the servant. The ORB uses this information to generate IORs and respond to **`_is_a()`** requests from clients. The method takes a POA instance and an **ObjectId** as an argument and returns a sequence of repository ids representing the type of information for that **`oid`**. The repository id at the zero index represents the most derived interface. The last id, for the generic CORBA Object (i.e., "IDL:omg.org/CORBA/Object:1.0"), is implied and not present. An implementor of this method must return complete type information for the specified **`oid`** for the ORB to behave correctly.

_non_existent

`Servant` provides a default implementation of **`_non_existent()`** that can be overridden by derived servants if the default behavior is not adequate.

_get_interface

`Servant` provides a default implementation of **`_get_interface()`** that can be overridden by derived servants if the default behavior is not adequate.

_is_a

`Servant` provides a default implementation of **`_is_a()`** that can be overridden by derived servants if the default behavior is not adequate. The default implementation checks to see if the specified **`repid`** is present on the list returned by **`_all_interfaces()`** (see "_all_interfaces" on page 1-92) or is the repository id for the generic CORBA Object. If so, then **`_is_a()`** returns **`true`**; otherwise, it returns **`false`**.

1.20.2.2 *Mapping of Dynamic Skeleton Interface*

This section contains the following information:

- Mapping of the Dynamic Skeleton Interface's **ServerRequest** to Java
- Mapping of the Portable Object Adapter's Dynamic Implementation Routine to Java

Mapping of ServerRequest

The **ServerRequest** interface maps to the following Java class:

```
// Java

package org.omg.CORBA;

public abstract class ServerRequest {

   /**
    * @deprecated use operation()
    */
   public String op_name() {
      return operation();
   }

   public String operation() {
      throw new org.omg.CORBA.NO_IMPLEMENT();
   }

   public abstract Context ctx();

   /**
    * @deprecated use arguments()
    */
   public void params(NVList parms) {
      arguments(parms);
   }

   public void arguments(NVList nv) {
      throw new org.omg.CORBA.NO_IMPLEMENT();
   }

   /**
    * @deprecated use set_result()
    */
   public void result(Any a) {
      set_result(a);
   }

   public void set_result(Any val) {
      throw new org.omg.CORBA.NO_IMPLEMENT();
   }
```

```
    /**
     * @deprecated use set_exception()
     */
    public void except(Any a) {
       throw new org.omg.CORBA.NO_IMPLEMENT();
    }

    public void set_exception(Any val) {
       throw new org.omg.CORBA.NO_IMPLEMENT();
    }
}
```

Note that several methods have been deprecated in CORBA 2.3 in favor of the current methods as defined in CORBA 2.2 and used in the current C++ mapping. Implementations using the POA should use the new routines.

Mapping of POA Dynamic Implementation Routine

In Java, POA-based DSI servants inherit from the standard **DynamicImplementation** class. This class inherits from the **Servant** class and is also defined in the **org.omg.PortableServer** package. The **DynamicImplementation** class is defined as follows:

```
// Java
package org.omg.PortableServer;

abstract public class DynamicImplementation extends Servant
{
   abstract public void invoke(org.omg.CORBA.ServerRequest
request);
}
```

The **invoke()** method receives requests issued to any CORBA object incarnated by the DSI servant and performs the processing necessary to execute the request.

The ORB user must also provide an implementation to the **_all_interfaces()** method declared by the **Servant** class.

The old deprecated version is specified below.

```
// Java
package org.omg.CORBA;

/**
*@deprecated org.omg.CORBA.DynamicImplementation
*/
public class DynamicImplementation
      extends org.omg.CORBA.portable.ObjectImpl {
/**
*@deprecated Deprecated by Portable Object Adapter
*/
```

```
   public void invoke(org.omg.CORBA.ServerRequest request {
      throw new org.omg.CORBA.NO_IMPLEMENT();
         }
}
```

1.20.2.3 Skeleton Portability

The Java language mapping defines a binary portability layer (see Section 1.21, "Java ORB Portability Interfaces," on page 1-100) in order to provide binary compatibility of servant implementations on ORBs from different vendors. The server-side mapping supports this by defining two models of code generations for skeletons: a stream-based model and a DSI-based model. The example in the rest of the server-side mapping section uses the DSI-based model.

1.20.2.4 Skeleton Operations

All skeleton classes provide a **_this()** method. The method provides equivalent behavior to calling **_this_object()** but returns the most derived Java interface type associated with the servant.

It should be noted that because of the way the inheritance hierarchy is set up, the Object returned by **_this()** is an instance of a stub as defined in Section 1.21.5, "Portability Stub and Skeleton Interfaces," on page 1-108.

1.20.2.5 Inheritance-Based Interface Implementation

Implementation classes can be derived from a generated base class based on the OMG IDL interface definition. The generated base classes are known as *skeleton classes*, and the derived classes are known as *implementation classes*. Each skeleton class implements the generated interface operations associated with the IDL interface definition. The implementation class shall provide implementations for each method defined in the interface operations class. It is important to note that the skeleton class does not extend the interface class associated with the IDL interface, but only implements the interface operations class.

For each IDL interface ***<interface_name>*** the mapping defines a Java class as follows:

```
// Java
import org.omg.PortableServer.POA;
import org.omg.PortableServer.DynamicImplementation;

abstract public class <interface_name>POA
            extends DynamicImplementation
            implements <interface_name>Operations {
   public <interface_name> _this() {
      return <interface_name>Helper.narrow(_this_object());
   }
   public <interface_name> _this(org.omg.CORBA.ORB orb) {
```

```
        return <interface_name>Helper.narrow(
                                    _this_object(orb));
    }
    public String[] _all_interfaces(
                        POA poa,
                        byte[] objectId)
        {...}
    public void invoke(
                        org.omg.CORBA.ServerRequest request)
        {...}
}
```

The implementation of **`_all_interfaces()`** and **`invoke()`** are provided by the compiler. The **`_all_interfaces()`** method must return the full type hierarchy as known at compile time.

For example, given the following IDL:

```
// IDL
interface A {
    short op1();
    void op2(in long val);
}
```

A skeleton class for interface **A** would be generated as follows:

```
// Java
import org.omg.PortableServer.POA;
import org.omg.PortableServer.DynamicImplementation;

abstract public class APOA extends DynamicImplementation
                        implements AOperations {
    public A _this() {
        return AHelper.narrow(_this_object());
    }
    public A _this(org.omg.CORBA.ORB orb) {
        return AHelper.narrow(_this_object(orb));
    }
    public String[] _all_interfaces(
                        POA poa,
                        byte[] objectId)
        {...}
    public void invoke(org.omg.CORBA.ServerRequest request)
        {...}
}
```

The user subclasses the **`APOA`** class to provide implementations for the methods on **`AOperations`**.

1.20.2.6 Delegation-Based Interface Implementation

Because Java does not allow multiple implementation inheritance, inheritance-based implementation is not always the best solution. Delegation can be used to help solve this problem. This section describes a delegation approach to implementation which is type safe.

For each IDL interface **<interface_name>** the mapping defines a Java tie class as follows:

```
//Java
import org.omg.PortableServer.POA;

public class <interface_name>POATie
            extends <interface_name>POA {
   private <interface_name>Operations _delegate;
   private POA _poa;

   public <interface_name>POATie(
                  <interface_name>Operations delegate) {
      _delegate = delegate;
   }
   public <interface_name>POATie(
                  <interface_name>Operations delegate,
                  POA poa) {
      _delegate = delegate;
      _poa = poa;
   }
   public <interface_name>Operations _delegate() {
      return _delegate;
   }
   public void _delegate(
                  <interface_name>Operations delegate) {
      _delegate = delegate;
   }
   public POA _default_POA() {
      if (_poa != null) {
         return _poa;
      }
      else {
         return super._default_POA();
      }
   }
   // for each method <method> defined in
<interface_name>Operations
   // The return statement is present for methods with
   //    return values.
   public <method> {
      [return] _delegate.<method>;
   }
}
```

Using the example above, a tie class for interface **A** would be generated as follows:

```
// Java
import org.omg.PortableServer.POA;

public class APOATie extends APOA {
   private AOperations _delegate;
   private POA _poa;
   public APOATie(AOperations delegate) {
      _delegate = delegate;
   }
   public APOATie(AOperations delegate, POA poa) {
      _delegate = delegate;
      _poa = poa;
   }
   public AOperations _delegate() {
      return _delegate;
   }
   public void _delegate(AOperations delegate) {
      _delegate = delegate;
   }
   public POA _default_POA() {
      if (_poa != null) {
         return _poa;
      }
      else {
         return super._default_POA();
      }
   }
   public short op1() {
      return _delegate.op1();
   }
   public void op2(int val) {
      _delegate.op2(val);
   }
   ...
}
```

To implement an interface using the delegation approach, a developer must write an implementation class, which implements the operations class associated with the interface they wish to implement. The developer then instantiates an instance of the implementation class and uses this instance in the constructor of the tie class associated with the interface. The tie class can then be used as servant in POA operations.

It is important to note that the implementation class has no access to the object reference associated with the tie object. One way for the delegate to access this information is for the delegate to keep a reference to the tie object. For a delegate which is tied to multiple tie objects, this approach will not work. Instead, the delegate can determine its current object reference by calling

PortableServer::Current::get_object_id() and passing the return value to **PortableServer::POA::id_to_reference()**. The result may then be narrowed to the appropriate interface if required.

1.20.3 Mapping for PortableServer::ServantManager

1.20.3.1 Mapping for Cookie

The native type **PortableServer::ServantLocator::Cookie** is mapped to `java.lang.Object`. A `CookieHolder` class is provided for passing the **Cookie** type as an *out* parameter. The `CookieHolder` class follows exactly the same pattern as the other holder classes for basic types. See Section 1.4.1.4, "Holder Classes," on page 1-7 for details. The class is defined as follows:

```
package org.omg.PortableServer.ServantLocatorPackage;

final public class CookieHolder implements
org.omg.CORBA.portable.Streamable {
   public java.lang.Object value;
   public CookieHolder() {...}
   public CookieHolder(java.lang.Object initial) {...}
   public void _read(
            org.omg.CORBA.portable.InputStream is)
      {...}
   public void _write(
            org.omg.CORBA.portable.OutputStream os)
      {...}
   public org.omg.CORBA.TypeCode _type() {...}
}
```

For the Java mapping of the **PortableServer::ServantLocator::preinvoke()** operation, a `CookieHolder` object will be passed in with its value field set to `null`, the user may then set the value to any Java object. The same `Cookie` object will then be passed to the **PortableServer::ServantLocator::postinvoke()** operation.

1.20.3.2 ServantManagers and AdapterActivators

Portable servants that implement the **PortableServer::AdapterActivator**, the **PortableServer::ServantActivator**, or **PortableServer::ServantLocator** interfaces are implemented just like any other servant. They may use either the inheritance-based approach or the delegation-based approach.

1.21 Java ORB Portability Interfaces

1.21.1 Introduction

The APIs specified here provide the minimal set of functionality to allow portable stubs and skeletons to be used with a Java ORB. The interoperability requirements for Java go beyond that of other languages. Because Java classes are often downloaded and come from sources that are independent of the ORB in which they will be used, it is essential to define the interfaces that the stubs and skeletons use. Otherwise, use of a stub (or skeleton) will require: either that it have been generated by a tool that was provided by the ORB vendor (or is compatible with the ORB being used), or that the entire ORB runtime be downloaded with the stub or skeleton. Both of these scenarios are unacceptable.

Two such styles of interfaces are defined, one based on the DII/DSI, the other based on a streaming approach. Conforming ORB Java runtimes shall support both styles. A conforming vendor tool may choose between the two styles of stubs/skeletons to generate, but shall support the generation of at least one style.

1.21.1.1 Design Goals

The design balances several goals:

- Size - Stubs and skeletons must have a small bytecode footprint in order to make downloading fast in a browser environment and to minimize memory requirements when bundled with a Java VM, particularly in specialized environments such as set-top boxes.
- Performance - Obviously, the runtime performance of the generated stub code must be excellent. In particular, care must be taken to minimize temporary Java object creation during invocations in order to avoid Java VM garbage collection overhead.

A very simple delegation scheme is specified here. Basically, it allows ORB vendors maximum flexibility for their ORB interfaces, as long as they implement the interface APIs. Of course vendors are free to add proprietary extensions to their ORB runtimes. Stubs and skeletons which require proprietary extensions will not necessarily be portable or interoperable and may require download of the corresponding runtime.

1.21.2 Overall Architecture

The stub and skeleton portability architecture supports the use of both the DII/DSI, and a streaming API as its portability layer. The mapping of the DII and DSI PIDL have operations that support the efficient implementation of portable stubs and skeletons.

The major components to the architecture are:

- Portable Streamable - provides standard APIs to read and write IDL datatypes
- Portable Streams - provide standard APIs to the ORB's marshaling engine

- Portable Stubs and Skeletons - provides standard APIs that are used to connect stubs and skeletons with the ORB
- Portable Delegate - provides the vendor specific implementation of CORBA object
- Portable Servant Delegate - provides the vendor specific implementation of **PortableServer::Servant**
- ORB Initialization - provides standard way to initialize the ORB

1.21.2.1 Portability Package

The APIs needed to implement portability are found in the **org.omg.CORBA.portable** and **org.omg.PortableServer.portable** package.

The portability package contains interfaces and classes that are designed for and intended to be used by ORB implementor. It exposes the publicly defined APIs that are used to connect stubs and skeletons to the ORB.

1.21.3 Streamable APIs

The Streamable Interface API provides the support for the reading and writing of complex data types. It is implemented by static methods on the Helper classes. They are also used in the Holder classes for reading and writing complex data types passed as out and inout parameters.

```
package org.omg.CORBA.portable;

public interface Streamable {
   void _read(org.omg.CORBA.portable.InputStream is);
   void _write(org.omg.CORBA.portable.OutputStream os);
   org.omg.CORBA.TypeCode _type();
}
```

1.21.4 Streaming APIs

The streaming APIs are Java interfaces that provide for the reading and writing of all of the mapped IDL types to and from streams. Their implementations are used inside the ORB to marshal parameters and to insert and extract complex datatypes into and from **Any**s.

The streaming APIs are found in the **org.omg.CORBA.portable** and **org.omg.CORBA_2_3.portable** packages. The ORB object is used as a factory to create an output stream. An input stream may be created from an output stream.

```
package org.omg.CORBA;

interface ORB {
      OutputStream   create_output_stream();
};
```

```
package org.omg.CORBA.portable;

public abstract class InputStream
            extends java.io.InputStream {

   public int read() throws java.io.IOException {
      throw new org.omg.CORBA.NO_IMPLEMENT();
      }

   public org.omg.CORBA.ORB orb() {
      throw new org.omg.CORBA.NO_IMPLEMENT();
   }

   public abstract booleanread_boolean();

   public abstract charread_char();

   public abstract charread_wchar();

   public abstract byteread_octet();

   public abstract shortread_short();

   public abstract shortread_ushort();

   public abstract intread_long();

   public abstract intread_ulong();

   public abstract longread_longlong();

   public abstract longread_ulonglong();

   public abstract floatread_float();

   public abstract doubleread_double();

   public abstract Stringread_string();

   public abstract Stringread_wstring();

   public abstract voidread_boolean_array(boolean[] value,
                                 int offset, int length);

   public abstract voidread_char_array(char[] value,
                                 int offset, int length);

   public abstract voidread_wchar_array(char[] value,
                                 int offset, int length);

   public abstract voidread_octet_array(byte[] value,
```

```
                                    int offset, int length);

public abstract voidread_short_array(short[] value,
                                    int offset, int length);

public abstract voidread_ushort_array(short[] value,
                                    int offset, int length);

public abstract voidread_long_array(int[] value,
                                    int offset, int length);

public abstract voidread_ulong_array(int[] value,
                                    int offset, int length);

public abstract voidread_longlong_array(long[] value,
                                    int offset, int length);

public abstract voidread_ulonglong_array(long[] value,
                                    int offset, int length);

public abstract voidread_float_array(float[] value,
                                    int offset, int length);

public abstract voidread_double_array(double[] value,
                                    int offset, int length);

public abstract org.omg.CORBA.Object   read_Object();

public org.omg.CORBA.Object            read_Object(
                                    java.lang.Class clz) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

public abstract org.omg.CORBA.TypeCode read_TypeCode();

public abstract org.omg.CORBA.Any   read_any();

public org.omg.CORBA.Context        read_Context() {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

/**
*@deprecated Deprecated by CORBA 2.2.
*/
public org.omg.CORBA.Principal      read_Principal() {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

public java.math.BigDecimal         read_fixed() {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

```
}

public abstract class OutputStream
         extends java.io.OutputStream {

   public void write(int b) throws java.io.IOException {
      throw new org.omg.CORBA.NO_IMPLEMENT();
      }

   public org.omg.CORBA.ORB orb() {
      throw new org.omg.CORBA.NO_IMPLEMENT();
   }

   public abstract InputStream       create_input_stream();

   public abstract void write_boolean  (boolean value);

   public abstract void write_char     (char value);

   public abstract void write_wchar    (char value);

   public abstract void write_octet    (byte value);

   public abstract void write_short    (short value);

   public abstract void write_ushort   (short value);

   public abstract void write_long     (int value);

   public abstract void write_ulong    (int value);

   public abstract void write_longlong (long value);

   public abstract void write_ulonglong(long value);
   public abstract void write_float    (float value);

   public abstract void write_double   (double value);

   public abstract void write_string   (String value);

   public abstract void write_wstring  (String value);

   public abstract void write_boolean_array(boolean[] value,
                               int offset, int length);

   public abstract void write_char_array  (char[] value,
                               int offset, int length);

   public abstract void write_wchar_array (char[] value,
                               int offset, int length);
```

```
public abstract void write_octet_array (byte[] value,
                            int offset, int length);

public abstract void write_short_array (short[] value,
                            int offset, int length);

public abstract void write_ushort_array   (short[] value,
                            int offset, int length);

public abstract void write_long_array     (int[] value,
                            int offset, int length);

public abstract void write_ulong_array    (int[] value,
                            int offset, int length);

public abstract void write_longlong_array (long[] value,
                            int offset, int length);

public abstract void write_ulonglong_array(long[] value,
                            int offset, int length);

public abstract void write_float_array    (float[] value,
                            int offset, int length);

public abstract void write_double_array(double[] value,
                            int offset, int length);

public abstract void write_Object(
                           org.omg.CORBA.Object value);

public abstract void write_TypeCode(
                           org.omg.CORBA.TypeCode value);

public abstract void write_any (org.omg.CORBA.Any value);

public void write_Context(
                  org.omg.CORBA.Context ctx,
                  org.omg.CORBA.ContextLists contexts) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

/**
*@deprecated Deprecated by CORBA 2.2.
*/
public void write_Principal(
                  org.omg.CORBA.Principal value) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

public void write_fixed(java.math.BigDecimal value {
   throw new org.omg.CORBA.NO_IMPLEMENT();
```

```
    }
}

package org.omg.CORBA_2_3.portable;

public abstract class InputStream
            extends org.omg.CORBA.portable.InputStream {

    public java.io.Serializable read_value() {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public java.io.Serializable read_value(
                        java.lang.String rep_id) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public java.io.Serializable read_value(Class clz) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public java.io.Serializable read_value(
                    org.omg.CORBA.BoxedValueHelper factory) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public java.io.Serializable read_value(
                    java.io.Serializable value) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public java.lang.Object read_abstract_interface() {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public java.lang.Object read_abstract_interface(
                    java.lang.Class clz) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
}

public abstract class OutputStream
         extends org.omg.CORBA.portable.OutputStream {

    public void write_value(
                    java.io.Serializable value) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public void write_value(
                    java.io.Serializable value,
                    java.lang.String rep_id) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public void write_value(
                    java.io.Serializable value,
                    Class clz) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
```

```
    }
    public void write_value(
                    java.io.Serializable value,
                    org.omg.CORBA.BoxedValueHelper factory) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public void write_abstract_interface(
                    java.lang.Object obj) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
}
```

1.21.4.1 InputStream Method Semantics

read_Context

The **read_Context()** method reads a **Context** from the stream. The **Context** is read from the stream as a sequence of strings as specified in *The Common Object Request Broker: Architecture and Specification*, GIOP chapter.

read_Object

For **read_Object**, the **clz** argument is one of the following:

- the **Class** object for the stub class which corresponds to the type that is statically expected. Typically, the ORB runtime will allocate and return a stub object of this stub class.
- the **Class** object for the RMI/IDL interface type that is statically expected. The ORB runtime must allocate and return a stub object that conforms to this interface.

read_abstract_interface

For **read_abstract_interface**, the ORB runtime will return either a value object or a suitable stub object. The specified clz argument is one of the following:

- the **Class** object for the stub class which corresponds to the type that is statically expected.
- the **Class** object for the RMI/IDL interface type that is statically expected. If a stub object is returned, it must conform to this interface.

The **read_abstract_interface()** and **read_abstract_interface(clz)** actual implementations may throw the **org.omg.CORBA.portable.IndirectionException** exception.

read_value

The **read_value()** methods unmarshals a value type from the input stream. The specified **clz** is the declared type of the value to be unmarshaled. The specified **rep_id** identifies the type of the value type to be unmarshaled. The specified **factory** is the instance of the helper to be used for unmarshaling the boxed value.

The specified **value** is an uninitialized value that is added to the orb's indirection table before calling **Streamable._read()** or **CustomMarshal.unmarshal()** to unmarshal the value.

The **read_value()** and **read_value(clz)** actual implementations may throw the **org.omg.CORBA.portable.IndirectionException** exception.

1.21.4.2 OutputStream Method Semantics

create_input_stream

The **create_input_stream()** method returns a new input stream from the output stream. The method implements copy semantics, so that the current contents of the output stream is copied to the input stream. Anything subsequently written to the output stream is not visible to the newly created input stream.

write_Context

The **write_context()** method writes the specified **Context** to the stream. The **Context** is marshaled as a sequence of strings as specified in *The Common Object Request Broker: Architecture and Specification*, GIOP chapter. Only those Context values specified in the **contexts** parameter are actually written.

write_value

The **write_value()** methods marshals a value type to the output stream. The first parameter is the actual value to write. The specified **clz** is the declared type of the value to be marshaled. The specified **rep_id** identifies the type of the value type to be marshaled. The specific **factory** is the instance of the helper to be used for marshaling the boxed value.

1.21.5 Portability Stub and Skeleton Interfaces

1.21.5.1 Stub/Skeleton Architecture

The mapping defines a single stub that may be used for both local and remote invocation. Local invocation provides higher performance for collocated calls on Servants located in the same process as the client. Local invocation is also required for certain IDL types, which contain parameter types that cannot be marshaled remotely. Remote invocation is used to invoke operations on objects that are located in an address space separate from the client.

While a stub is using local invocation it provides complete location transparency. To provide the correct semantics, compliant programs comply with the parameter passing semantics defined in Section 1.12.2, “Parameter Passing Modes,” on page 1-32. When using local invocation the stub copies all valuetypes passed to them, either as in parameters, or as data within in parameters, and passes the resulting copies to the Servant in place of the originals. The valuetypes are copied using the same deep copy semantics as would result from GIOP marshaling and unmarshaling.

The following sections describe the characteristics of the stubs and skeletons. The examples are based on the following IDL:

```
// Example IDL

module Example {

    exception AnException {};

    interface AnInterface {
        long length(in string s) raises (AnException);
};
```

Stub Design

All stubs inherit from a common base class **`org.omg.CORBA.portable.ObjectImpl`**. The class is responsible for delegating shared functionality such as **`is_a()`** to the vendor specific implementation. This model provides for a variety of vendor dependent implementation choices, while reducing the client-side and server "code bloat."

The stub is named ***_<interface_name>*`Stub`** where ***<interface_name>*** is the IDL interface name this stub is implementing and implements the signature interface ***<interface_name>***. Stubs support both local invocation and remote invocation, except in the following case:

- The stub is implementing an IDL interface that may only be invoked locally (e.g., **PortableServer::POA**). In this case, the stub may choose to implement only local invocation.
- The stub class supports either the DII or the streaming style APIs.

Skeleton Design

Skeletons may be either stream-based or DSI-based.

Stream-based skeletons directly extend the **`org.omg.PortableServer.Servant`** class (Section 1.20.2.1, "Mapping of PortableServer::Servant," on page 1-88) and implement the **`InvokeHandler`** interface (Section 1.21.5.4, "Invoke Handler," on page 1-122) as well as the operations interface associated with the IDL interface the skeleton implements.

DSI-based skeletons directly extend the **`org.omg.PortableServer.DynamicImplementation`** class (Section 1.20.2.2, "Mapping of Dynamic Skeleton Interface," on page 1-93) and implement the operations interface associated with the IDL interface the skeleton implements.

Stream-based Stub Example

```
package Example;
```

```
public class _AnInterfaceStub
            extends org.omg.CORBA.portable.ObjectImpl
            implements AnInterface {
   public java.lang.String[] _ids () {
      return __ids;
   }

   private static java.lang.String[] __ids =
                        { "IDL:Example/AnInterface:1.0" };

   final public static java.lang.Class _opsClass =
      Example.AnInterfaceOperations.class;

   public int length(java.lang.String s)
               throws Example.AnException {
      while(true) {
         if(!this._is_local()) {
            org.omg.CORBA.portable.OutputStream output =
                                                   null;
            org.omg.CORBA.portable.InputStream input = null;
            try {
               _output = this._request("length", true);
               _output.write_string(s);
               _input = this._invoke(_output);
               return _input.read_long();
               }
            catch (
   org.omg.CORBA.portable.RemarshalException _exception){
            continue;
               }
            catch (
   org.omg.CORBA.portable.ApplicationException _exception){
               java.lang.String _exception_id =
                     _exception.getId();
               if (_exception_id.equals(
                        Example.AnExceptionHelper.id())) {
                   _input = _exception.getInputStream();
                  throw Example.AnExceptionHelper.read(
                                                  _input);
               }
               throw new org.omg.CORBA.UNKNOWN(
            "Unexpected User Exception: " +_exception_id);
               }
               finally {
               this._releaseReply(_input);
               }
            }
         else {
            org.omg.CORBA.portable.ServantObject _so =
               _servant_preinvoke("length", _opsClass);
            if (_so == null) {
```

```
                  continue;
               }
               Example.AnInterfaceOperations _self =
                  (Example.AnInterfaceOperations)_so.servant;
               try {
                  return _self.length(s);
               }
               finally {
                  _servant_postinvoke(_so);
               }
            }
         }
      }
   }
```

Stream-based Skeleton Example

```
package Example;

abstract public class AnInterfacePOA
   extends org.omg.PortableServer.Servant
   implements org.omg.CORBA.portable.InvokeHandler,
            Example.AnInterfaceOperations {

   public Example.AnInterface _this() {
      return Example.AnInterfaceHelper.narrow(
                                 super._this_object());
   }
   public Example.AnInterface _this(org.omg.CORBA.ORB orb) {
      return Example.AnInterfaceHelper.narrow(
                                 super._this_object(orb));
   }
   public java.lang.String[] _all_interfaces(
                        org.omg.PortableServer.POA poa,
                        byte[] objectId) {
      return __ids;
   }
   private static java.lang.String[] __ids = {
      "IDL:Example/AnInterface:1.0"
   };
   public org.omg.CORBA.portable.OutputStream _invoke(
      String opName,
      org.omg.CORBA.portable.InputStream is,
      org.omg.CORBA.portable.ResponseHandler handler) {
      org.omg.CORBA.portable.OutputStream _output = null;
      if ("length".equals(opName)) {
         try {
            java.lang.String s;
            s = is.read_string();
            int _result = this.length(s);
            _output = handler.createReply();
```

```
                _output.write_long(_result);
             }
             catch (Example.AnException _exception) {
                _output = handler.createExceptionReply();
                Example.AnExceptionHelper.write(
                                        _output, _exception);
             }
             return _output;
          }
          else {
             throw new org.omg.CORBA.BAD_OPERATION();
          }
       }
    }

    public class AnInterfacePOATie
             extends Example.AnInterfacePOA {

       private Example.AnInterfaceOperations _delegate;
       private org.omg.PortableServer.POA _poa;
       public AnInterfacePOATie(
                     Example.AnInterfaceOperations delegate) {
          this._delegate = delegate;
       }
       public AnInterfacePOATie(
                     Example.AnInterfaceOperations delegate,
          org.omg.PortableServer.POA poa) {
          this._delegate = delegate;
          this._poa = poa;
       }
       public Example.AnInterfaceOperations _delegate() {
          return this._delegate;
       }
       public void _delegate(
                     Example.AnInterfaceOperations delegate) {
          this._delegate = delegate;
       }
       public org.omg.PortableServer.POA _default_POA() {
          if(_poa != null) {
             return _poa;
          }
          else {
             return super._default_POA();
          }
       }
       public int length (java.lang.String s)
                throws Example.AnException {
          return this._delegate.length(s);
       }

    }
```

Dynamic (DII-based) Stub Example

```
package Example;

public class _AnInterfaceStub
      extends org.omg.CORBA.portable.ObjectImpl
      implements Example.AnInterface {
   public java.lang.String[] _ids() {
      return __ids;
   }
   private static java.lang.String[] __ids = {
      "IDL:Example/AnInterface:1.0"
   };

   public int length (java.lang.String s)
         throws Example.AnException {
      org.omg.CORBA.Request _request =
                        this._request("length");
      _request.set_return_type(
      _orb().get_primitive_tc(
                     org.omg.CORBA.TCKind.tk_long));
      org.omg.CORBA.Any $s = _request.add_in_arg();
      $s.insert_string(s);
      _request.exceptions().add(
                  Example.AnExceptionHelper.type());
      _request.invoke();
      java.lang.Exception _exception =
                        _request.env().exception();
      if(_exception != null) {
         if(_exception instanceof
                  org.omg.CORBA.UnknownUserException) {
            org.omg.CORBA.UnknownUserException
               _userException =
            (org.omg.CORBA.UnknownUserException) _exception;
            if(_userException.except.type().equals(
                  Example.AnExceptionHelper.type())) {
               throw Example.AnExceptionHelper.extract(
                        _userException.except);
            }
            else {
               throw new org.omg.CORBA.UNKNOWN();
            }
         }
         throw (org.omg.CORBA.SystemException) _exception;
      }
      int _result;
      _result = _request.return_value().extract_long();
      return _result;
   }
```

```
}
```

Dynamic (DSI-based) Skeleton Example

```
package Example;

abstract public class AnInterfacePOA
      extends org.omg.PortableServer.DynamicImplementation
      implements Example.AnInterfaceOperations {
   public Example.AnInterface _this() {
      return Example.AnInterfaceHelper.narrow(
                           super._this_object());
   }
   public Example.AnInterface _this(org.omg.CORBA.ORB orb) {
      return Example.AnInterfaceHelper.narrow(
                           super._this_object(orb));
   }
   public java.lang.String[] _all_interfaces(
                           org.omg.PortableServer.POA poa,
                           byte[] objectId) {
      return __ids;
   }
   private static java.lang.String[] __ids = {
      "IDL:Example/AnInterface:1.0"
   };

   public void invoke(org.omg.CORBA.ServerRequest _request){
      java.lang.String _method = _request.operation();
      if("length".equals(_method)) {
         try {
            org.omg.CORBA.NVList _params =
                           _orb().create_list(1);
            org.omg.CORBA.Any $s = _orb().create_any();
            $s.type(_orb().get_primitive_tc(
                           org.omg.CORBA.TCKind.tk_string));
            _params.add_value(
                           "s",
                           $s,
                           org.omg.CORBA.ARG_IN.value);
            _request.arguments(_params);
            java.lang.String s;
            s = $s.extract_string();
            int _result = this.length(s);
            org.omg.CORBA.Any _resultAny =
                           _orb().create_any();
            _resultAny.insert_long(_result);
            _request.set_result(_resultAny);
         }
         catch (Example.AnException _exception) {
            org.omg.CORBA.Any _exceptionAny =
                           _orb().create_any();
```

```
                Example.AnExceptionHelper.insert(
                                  _exceptionAny, _exception);
                _request.set_exception(_exceptionAny);
            }
            return;
        }
        else {
            throw new org.omg.CORBA.BAD_OPERATION();
        }
    }
}
```

1.21.5.2 *Stub and Skeleton Class Hierarchy*

The required class hierarchy is shown in Figure 1-2 on page 1-116. The hierarchy is shown for a sample IDL interface **Foo**. Classes which are Java interfaces are indicated with the word *interface* before the class name. Classes in the **org.omg** package are defined by the Java mapping. Classes with a slash in the upper left-hand corner indicate classes that are generated by the IDL compiler or other tools. Classes beginning with **User** indicate user defined classes, which implement interfaces.

The following diagram shows the hierarchy used for DSI-based skeletons. For stream-based skeletons, the **org.omg.PortableServer.DynamicImplementation** class is omitted from the hierarchy, and **FooPOA** extends **org.omg.PortableServer.Servant** and implements **org.omg.CORBA.portable.InvokeHandler**.

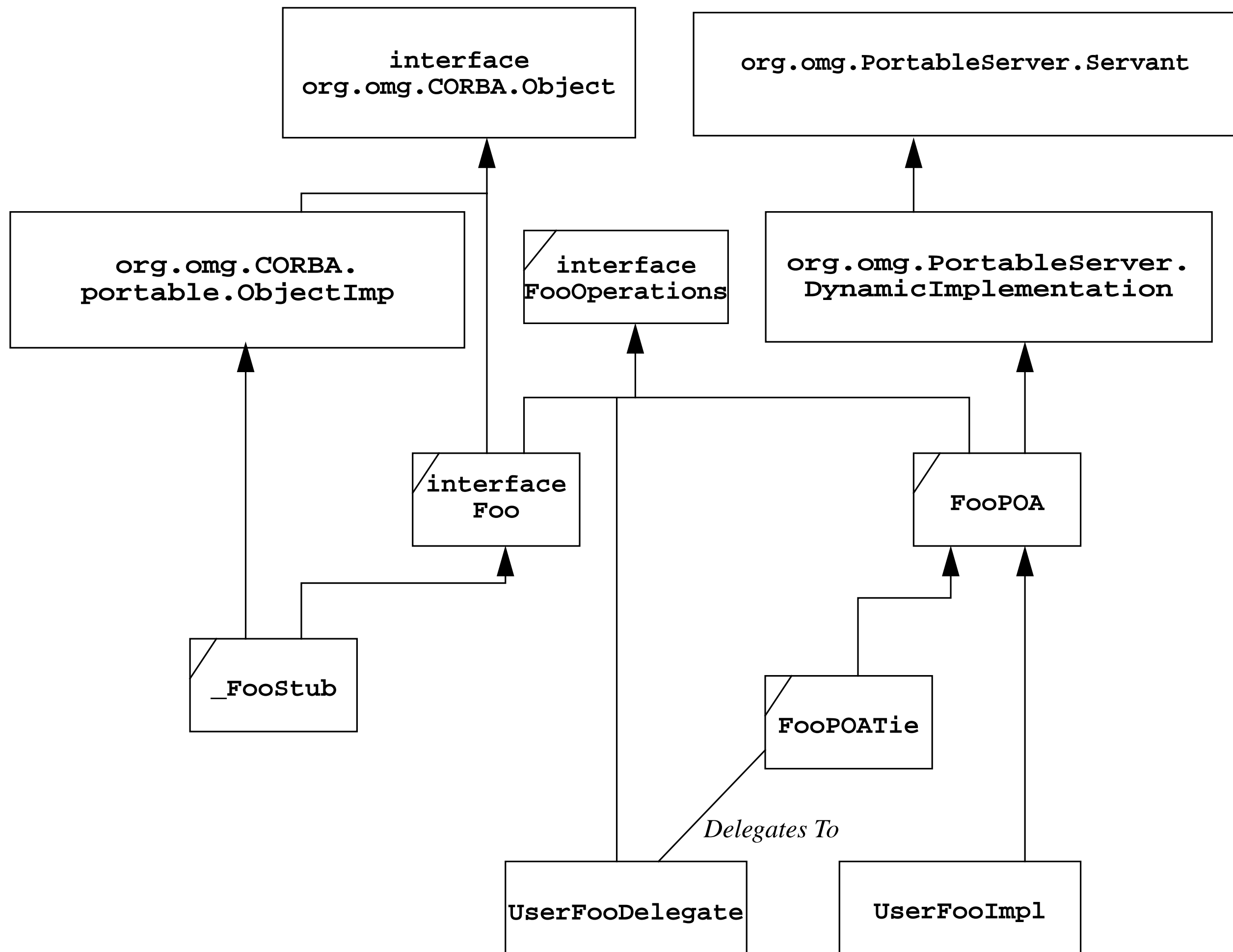

Figure 1-2 Class hierarchy for portable Java stubs and skeletons

1.21.5.3 *Portable ObjectImpl*

The ObjectImpl class is the base class for stubs. It provides the basic delegation mechanism.

```
package org.omg.CORBA.portable;

public class ServantObject {
   public java.lang.Object servant;
}

abstract public class ObjectImpl implements
                        org.omg.CORBA.Object {

   private transient Delegate __delegate;
```

```
   public Delegate _get_delegate() {
      if (__delegate == null) {
         throw new org.omg.CORBA.BAD_OPERATION();
      }
      return __delegate;
   }

   public void _set_delegate(Delegate delegate) {
      __delegate = delegate;
   }
   public abstract String[] _ids() {...}

// methods for standard CORBA stuff
   /**
   *@deprecated Deprecated by CORBA 2.3.
   */
   public org.omg.CORBA.InterfaceDef
                                 _get_interface() {
      return _get_delegate().get_interface(this);
   }

   public org.omg.CORBA.Object
                                 _get_interface_def() {
      return _get_delegate().get_interface_def(this);
   }

   public org.omg.CORBA.Object _duplicate() {
      return _get_delegate().duplicate(this);
   }

   public void _release() {
      _get_delegate().release(this);
   }

   public boolean _is_a(String repository_id) {
      return _get_delegate().is_a(this, repository_id);
   }

   public boolean _is_equivalent(org.omg.CORBA.Object rhs) {
      return _get_delegate().is_equivalent(this, rhs);
   }

   public boolean _non_existent() {
      return _get_delegate().non_existent(this);
   }

   public int _hash(int maximum) {
      return _get_delegate().hash(this, maximum);
   }
```

```
public org.omg.CORBA.Request _request(String operation) {
   return _get_delegate().request(this, operation);
}

public org.omg.CORBA.portable.OutputStream _request(
                              String operation,
                              boolean responseExpected) {
   return _get_delegate().request(
                           this,
                           operation,
                           responseExpected);
}

public org.omg.CORBA.portable.InputStream _invoke(
               org.omg.CORBA.portable.OutputStream os)
      throws ApplicationException, RemarshalException {
   return _get_delegate().invoke(this, os);
}

public void _releaseReply(
               org.omg.CORBA.portable.InputStream is) {
    _get_delegate().releaseReply(this, is);
}

public org.omg.CORBA.Request _create_request(
               org.omg.CORBA.Context ctx,
               String operation,
               org.omg.CORBA.NVList arg_list,
               org.omg.CORBA.NamedValue result) {
   return _get_delegate().create_request(
               this,
               ctx,
               operation,
               arg_list,
               result);
}

public Request _create_request(
               org.omg.CORBA.Context ctx,
               String operation,
               org.omg.CORBA.NVList arg_list,
               org.omg.CORBA.NamedValue result,
               org.omg.CORBA.ExceptionList exceptions,
               org.omg.CORBA.ContextList contexts) {
   return _get_delegate().create_request(
               this,
               ctx,
               operation,
               arg_list,
               result,
               exceptions,
```

```
                contexts);
}

public Policy _get_policy(int policy_type) {
   return _get_delegate().get_policy(this, policy_type);
}

public DomainManager[] _get_domain_managers() {
   return _get_delegate().get_domain_managers(this);
}

public org.omg.CORBA.Object _set_policy_override(
                org.omg.CORBA.Policy [] policies,
                org.omg.CORBA.SetOverrideType set_add) {
      return _get_delegate().set_policy_override(
                this,
                policies,
                set_add);
}

public org.omg.CORBA.ORB _orb() {
   return _get_delegate().orb(this);
}

public boolean _is_local() {
   return _get_delegate().is_local(this);
}

public ServantObject _servant_preinvoke(
                      String operation,
                      Class expectedType) {
   return _get_delegate().servant_preinvoke(
                      this,
                      operation,
                      expectedType);
}

public void _servant_postinvoke(ServantObject servant) {
   _get_delegate().servant_postinvoke(this, servant);
}

public String toString() {
   if ( __delegate != null )
      return __delegate.toString(this);
   else
      return getClass().getName() + ": no delegate set";
}

public int hashCode() {
   if ( __delegate != null )
      return __delegate.hashCode(this);
```

```
        else
            return System.identityHashCode(this);
    }

    public boolean equals(java.lang.Obecjt obj) {
        if ( __delegate != null )
            return __delegate.equals(this, obj);
        else
            return (this==obj);
    }
}

package org.omg.CORBA_2_3.portable;

public abstract class ObjectImpl extends
                org.omg.CORBA.portable.ObjectImpl {
    /** Returns the codebase for this object reference.
     * @return the codebase as a space delimited list of url
     * strings or null if none
    */
    public java.lang.String _get_codebase() {
        org.omg.CORBA.portable.Delegate delegate =
                        _get_delegate();
        if (delegate instanceof
                        org.omg.CORBA_2_3.portable.Delegate)
            return ((org.omg.CORBA_2_3.portable.Delegate)
                        delegate).get_codebase(this);
        return null;
    }
}
```

_ids

The method **_ids()** returns an array of repository ids that an object implements. The string at the zero index represents the most derived interface. The last id, for the generic CORBA object (i.e., "IDL:omg.org/CORBA/Object:1.0") is implied and not present.

Streaming Stub APIs

The method **_request()** is called by a stub to obtain an **OutputStream** for marshaling arguments. The stub must supply the operation name, and indicate if a response is expected (i.e., is this a one way call).

The method **_invoke()** is called to invoke an operation. The stub provides an OutputStream that was previously returned from a **_request()** call. The method **_invoke()** returns an **InputStream** that contains the marshaled reply. The **_invoke()** method may throw only one of the following: an **ApplicationException**, a **RemarshalException**, or a CORBA system exception as described below:

- An **ApplicationException** is thrown to indicate the target has raised a CORBA user exception during the invocation. The stub may access the **InputStream** of the **ApplicationException** to unmarshal the exception data.
- A **RemarshalException** is thrown if the stub was redirected to a different target object and remarshaling is necessary, this is normally due to a GIOP object forward or locate forward message. In this case, the stub then attempts to reinvoke the request on behalf of the client after verifying the target is still remote by invoking **_is_local()** (see "Local Invocation APIs" on page 1-121). If **_is_local()** returns **True**, then an attempt to reinvoke the request using the Local Invocation APIs shall be made.
- If the CORBA system exception org.omg.CORBA.portable.UnknownException is thrown, then the stub does one of the following:
  - Translates it to **org.omg.CORBA.UNKNOWN**.
  - Translates it to the nested exception that the UnknownException contains.
  - Passes it on directly to the user.
- If the CORBA system exception being thrown is not org.omg.CORBA.portable.UnknownException, then the stub passes the exception directly to the user.

The method **_releaseReply()** may optionally be called by a stub to release a reply stream back to the ORB when unmarshaling has completed. The stub passes the InputStream returned by **_invoke()** or **ApplicationException.getInputStream()**. A null value may also be passed to **_releaseReply()**, in which case the method is a noop. This method may be used by the ORB to assist in buffer management.

Local Invocation APIs

Local invocation is supported by the following methods and classes.

The **_is_local()** method is provided so that stubs may determine if a particular object is implemented by a local servant and hence local invocation API's may be used. The **_is_local()** method returns true if the servant incarnating the object is located in the same process as the stub and they both share the same ORB instance. The **_is_local()** method returns **false** otherwise. The default behavior of **_is_local()** is to return **false**.

The **_servant_preinvoke()** method is invoked by a local stub to obtain a Java reference to the servant that should be used for this request. The method takes a string containing the operation name and a Class object representing the expected type of the servant as parameters and returns a ServantObject object.

Note – ORB vendors may subclass the ServantObject object to return an additional request state that may be required by their implementations.

The operation name corresponds to the operation name as it would be encoded in a GIOP request. The expected type is the Class object associated with the operations class of the stub's interface (e.g., a stub for an interface **Foo**, would pass the Class object for the **FooOperations** interface). The method returns a **null** value if the servant is not local or the servant has ceased to be local as a result of the call (i.e., due to a ForwardRequest from a POA ServantManager). The method throws an org.omg.CORBA.BAD_PARAM exception if the servant is not of the expected type. If a **ServantObject** object is returned, then the servant field has been set to an object of the expected type.

Note – The object may or may not be the actual servant instance.

The local stub may cast the servant field to the expected type, and then invoke the operation directly. The **ServantRequest** object is valid for only one invocation, and cannot be used for more than one invocation.

The **_servant_postinvoke()** method is invoked after the operation has been invoked on the local servant. The local stub must pass the instance of the **ServerObject** object returned from the **_servant_preinvoke()** method as an argument. This method must be called if **_servant_preinvoke()** returned a non-null value, even if an exception was thrown by the servant's method. For this reason, the call to **_servant_postinvoke()** should be placed in a Java **finally** clause.

1.21.5.4 Invoke Handler

The **org.omg.CORBA.portable.InvokeHandler** interface provides a dispatching mechanism for an incoming call. It is invoked by the ORB to dispatch a request to a servant.

```
package org.omg.CORBA.portable;

public interface InvokeHandler {
        OutputStream _invoke(
                                        String method,
                                        InputStream is,
                                        ResponseHandler handler)
      throws org.omg.CORBA.SystemException;
}
```

The **_invoke()** method receives requests issued to any servant that implements the **InvokeHandler** interface. The **InputStream** contains the marshaled arguments. The specified **ResponseHandler** will be used by the servant to construct a proper reply. The only exceptions that may be thrown by this method are CORBA SystemExceptions. The returned **OutputStream** is created by the **ResponseHandler** and contains the marshaled reply.

A servant shall not retain a reference to the **ResponseHandler** beyond the lifetime of the method invocation.

Servant behavior is defined as follows:

- Determine correct method, and unmarshal parameters from **InputStream**
- Invoke method implementation.
- If no user exception, create a normal **Reply** using the **ResponseHandler**
- If user exception occurred, create an exception reply using **ResponseHandler**
- Marshal reply into **OutputStream** returned by the **ResponseHandler**
- Return the **OutputStream** to the ORB

1.21.5.5 Response Handler

The **org.omg.CORBA.portable.ResponseHandler** interface is supplied by an ORB to a servant at invocation time and allows the servant to later retrieve an **OutputStream** for returning the invocation results.

```
package org.omg.CORBA.portable;

public interface ResponseHandler {

    /**
      * Called by servant during a method invocation.
      * The servant should call
      * this method to create a reply marshal buffer if
      * no exception occurred.
      *
      * Returns an OutputStream suitable for
      * marshalling reply.
      */
    OutputStream createReply();

    /**
      * Called by servant during a method invocation.
      * The servant should call
      * this method to create a reply marshal buffer if
      * a user exception occurred.
      *
      * Returns an OutputStream suitable for marshalling
      * the exception ID and the user exception body.
      */
    OutputStream createExceptionReply();
}
```

1.21.5.6 Application Exception

The **org.omg.CORBA.portable.ApplicationException** class is used for reporting application level exceptions between ORBs and stubs.

The method **getId()** returns the CORBA repository ID of the exception without removing it from the exception's input stream.

```
package org.omg.CORBA.portable;
```

```
public class ApplicationException extends Exception {

   public ApplicationException(
                        String id,
                        org.omg.CORBA.portable.InputStream is)
      {...}

   public String getId() {...}

   public org.omg.CORBA.portable.InputStream
                                          getInputStream()
      {...}
}
```

The constructor takes the CORBA repository ID of the exception and an input stream from which the exception data can be read as its parameters.

1.21.5.7 Remarshal Exception

The **org.omg.CORBA.portable.RemarshalException** class is used for reporting locate forward exceptions and object forward GIOP messages back to the ORB. In this case the ORB must remarshal the request before trying again. See "Stub Design" on page 1-109 for more information.

```
package org.omg.CORBA.portable;

final public class RemarshalException extends Exception {
   public RemarshalException() {
      super();
   }
}
```

1.21.5.8 UnknownException

The **org.omg.CORBA.portable.UnknownException** is used for reporting unknown exceptions between ties and ORBs and between ORBs and stubs. It provides a Java representation of an UNKNOWN system exception that has an **UnknownExceptionInfo** service context.

```
package org.omg.CORBA.portable;

public class UnknownException extends
                        org.omg.CORBA.SystemException {
   public Throwable originalEx;
   public UnknownException(Throwable ex) {
      super("", 0, CompletionStatus.COMPLETED_MAYBE);
      originalEx = ex;
   }
}
```

1.21.6 Delegate Stub

The delegate class provides the ORB vendor specific implementation of CORBA object.

```
// Java

package org.omg.CORBA.portable;

public abstract class Delegate {
   /**
   *@deprecated Deprecated by CORBA 2.3.
   */
   public abstract org.omg.CORBA.InterfaceDef get_interface(
                        org.omg.CORBA.Object self);

   public abstract org.omg.CORBA.Object get_interface_def(
                        org.omg.CORBA.Object self);

   public abstract org.omg.CORBA.Object duplicate(
                        org.omg.CORBA.Object self);

   public abstract void release(org.omg.CORBA.Object self);

   public abstract boolean is_a(org.omg.CORBA.Object self,
                        String repository_id);

   public abstract boolean non_existent(
                        org.omg.CORBA.Object self);

   public abstract boolean is_equivalent(
                        org.omg.CORBA.Object self,
                        org.omg.CORBA.Object rhs);

   public abstract int hash(
                        org.omg.CORBA.Object self
                        int max);

   public abstract org.omg.CORBA.Request create_request(
                        org.omg.CORBA.Object self,
                        org.omg.CORBA.Context ctx,
                        String operation,
                        org.omg.CORBA.NVList arg_list,
                        org.omg.CORBA.NamedValue result);

   public abstract org.omg.CORBA.Request create_request(
                        org.omg.CORBA.Object self,
                        org.omg.CORBA.Context ctx,
                        String operation,
                        org.omg.CORBA.NVList arg_list,
                        org.omg.CORBA.NamedValue result,
```

```
                            org.omg.CORBA.ExceptionList excepts,
                            org.omg.CORBA.ContextList contexts);

public abstract org.omg.CORBA.Request request(
                            org.omg.CORBA.Object self,
                            String operation);

public org.omg.CORBA.portable.OutputStream request(
                            org.omg.CORBA.Object self,
                            String operation,
                            boolean responseExpected) {
   throw new org.omg.CORBA.NO_IMPLEMENT();

public org.omg.CORBA.portable.InputStream invoke(
                        org.omg.CORBA.Object self,
                        org.omg.CORBA.portable.OutputStream os)
      throws ApplicationException, RemarshalException {
   throw new org.omg.CORBA.NO_IMPLEMENT();

public void releaseReply(
                    org.omg.CORBA.Object self,
                    org.omg.CORBA.portable.InputStream is) {
   throw new org.omg.CORBA.NO_IMPLEMENT();

public org.omg.CORBA.Policy get_policy(
                    org.omg.CORBA.Object self,
                    int policy_type) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

public org.omg.CORBA.DomainManager[] get_domain_managers(
                    org.omg.CORBA.Object self) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

public org.omg.CORBA.Object set_policy_override(
                org.omg.CORBA.Object self,
                org.omg.CORBA.Policy[] policies,
                org.omg.CORBA.SetOverrideType set_add) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

public org.omg.CORBA ORB orb(
                            org.omg.CORBA.Object self) {
   throw new org.omg.CORBA.NO_IMPLEMENT();
}

public boolean is_local(org.omg.CORBA.Object self) {
   return false;
}
```

```
    public ServantObject servant_preinvoke(
                        org.omg.CORBA.Object self,
                        String operation, Class expectedType) {
       return null;
    }

    public void servant_postinvoke(
                        org.omg.CORBA.Object self,
                        ServantObject servant) {
    }

    public String toString(org.omg.CORBA.Object self) {
       return self.getClass().getName() + ":" +
                              this.toString();
    }

    public int hashCode(org.omg.CORBA.Object self) {
       return System.identityHashCode(self);
    }

    public boolean equals(
                        org.omg.CORBA.Object self,
                        java.lang.Object obj) {
       return (self==obj);
    }
}

package org.omg.CORBA_2_3.portable;

public abstract class Delegate extends
                        org.omg.CORBA.portable.Delegate {
    /** Returns the codebase for this object reference.
     * @param self the object reference for which to return
     * the codebase
     * @return the codebase as a space delimited list of url
     * strings or null if none
    */
    public java.lang.String get_codebase(
                        org.omg.CORBA.Object self) {
       return null;
    }
}
```

1.21.7 Servant

The Servant class is the base class for all POA-based implementations. It delegates all functionality to the **Delegate** interface defined in Section 1.21.6, “Delegate Stub,” on page 1-125.

Its specification can be found in Section 1.20.3, “Mapping for PortableServer::ServantManager,” on page 1-99.

```
package org.omg.PortableServer;

abstract public class Servant {
   ...
}
```

1.21.8 Servant Delegate

The Delegate interface provides the ORB vendor specific implementation of **PortableServer::Servant**.

```
package org.omg.PortableServer.portable;

import org.omg.PortableServer.Servant;
import org.omg.PortableServer.POA;

public interface Delegate {
   org.omg.CORBA.ORB orb(Servant self);
   org.omg.CORBA.Object this_object(Servant self);
   POA poa(Servant self);
   byte[] object_id(Servant self);
   POA default_POA(Servant self);
   boolean is_a(Servant self, String repository_id);
   boolean non_existent(Servant self);
   org.omg.CORBA.InterfaceDef get_interface(Servant self);
}
```

1.21.9 ORB Initialization

The ORB class represents an implementation of a CORBA ORB. Vendor specific ORB implementations can extend this class to add new features.

There are several cases to consider when creating the ORB instance. An important factor is whether an applet in a browser or an stand-alone Java application is being used.

In any event, when creating an ORB instance, the class names of the ORB implementation are located using the following search order:

- check in Applet parameter or application string array, if any
- check in properties parameter, if any
- check in the System properties
- check in orb.properties file, if it exists (Section 1.21.9.2, “orb.properties file,” on page 1-129)
- fall back on a hardcoded default behavior

1.21.9.1 Standard Properties

The OMG standard properties are defined in the following table.

Table 1-3 Standard ORB properties

Property Name	Property Value
org.omg.CORBA.ORBClass	class name of an ORB implementation
org.omg.CORBA.ORBSingletonClass	class name of the singleton ORB implementation

1.21.9.2 orb.properties file

The **orb.properties** file is an optional file located in the **<java-home>/lib** directory, where **<java-home>** is the value of the System property **java.home**. It consists of lines of the form **<property-name>=<property-value>.**

See Table 1-3 for a list of the property names and values that are recognized by ORB.init. Any property names not in this list shall be ignored by **ORB.init()**. The file may also contain blank lines and comment lines (starting with #), which are ignored.

1.21.9.3 ORB Initialization Methods

There are three forms of initialization as shown below. In addition the actual ORB implementation (subclassed from **ORB**) must implement the **set_parameters()** methods so that the initialization parameters will be passed into the ORB from the initialization methods.

```
// Java

package org.omg.CORBA;

abstract public class ORB {

   // Application init

   public static ORB init(String[] args,
                          java.util.Properties props) {
      // call to: set_parameters(args, props);
      ...
      }

   // Applet init

   public static ORB init(java.applet.Applet app,
                          java.util.Properties props) {
      // call to: set_parameters(app, props);
```

```
        ...
        }

    // Default (singleton) init

    public static ORB init()
        {...}

    // Implemented by subclassed ORB implementations
    // and called by init methods to pass in their params

    abstract protected void set_parameters(String[] args,
                                java.util.Properties props);
    abstract protected void set_parameters(Applet app,
                                java.util.Properties props);

}
```

Default initialization

The default initialization method returns the singleton ORB. If called multiple times it will always return the same the Java object.

The primary use of the no-argument version of **`ORB.init()`** is to provide a factory for **TypeCode**s for use by Helper classes implementing the **`type()`** method, and to create Any instances that are used to describe union labels as part of creating a union **TypeCode**. These Helper classes may be baked-in to the browser (e.g., for the interface repository stubs or other wildly popular IDL) and so may be shared across untrusted applets downloaded into the browser. The returned ORB instance is shared across all applets and therefore must have sharply restricted capabilities so that unrelated applets can be isolated from each other. It is not intended to be used directly by applets. Therefore, the ORB returned by **`ORB.init()`**, if called from a Java applet, may only be used to create **Typecode**s.

The following list of ORB methods are the only methods which may be called on the singleton ORB. An attempt to invoke any other ORB method shall raise the system exception NO_IMPLEMENT.

- **`create_xxx_tc(), where xxx is one the defined typecode types`**
- **`get_primitive_tc()`**
- **`create_any()`**

Application initialization

The application initialization method should be used from a stand-alone Java application. It is passed a array of strings which are the command arguments and a list of Java properties. Either the argument array or the properties may be **`null`**.

It returns a new fully functional ORB Java object each time it is called.

Applet initialization

The applet initialization method should be used from an applet. It is passed “the applet” and a list of Java properties. Either the applet or the properties may be **`null`**.

It returns a new fully functional ORB Java object each time it is called.