ORB Portability IDL/Java Language Mapping

Joint Revised Submission

BEA Systems, Inc.
Borland International, Inc.
IONA Technologies, Ltd.
International Business Machines Corporation

Supported by:

Sun Microsystems, Inc.

OMG TC Document orbos/98-03-10
(orbos/98-01-06 with errata)

March 13, 1998 3:58 pm

The submitting companies listsed above have all contributed to this “merged” submission. These companies recognize that this draft joint submission is the joint intellectual property of all the submitters, and may be used by any of them in the future, regardless of whether they ultimately participate in a final joint submission.

Preface 1

1.1 Cosubmitting Companies

The following companies are pleased to jointly submit this specification in response to the OMG Java Language Mapping RFP (Document orbos/96-08-01):

- BEA Systems, Inc.
- Borland International, Inc.
- International Business Machines Corporation
- IONA Technologies, Ltd.

1.2 Guide to the Submission

This submission specifies changes to the existing IDL/Java mapping as specified in CORBA 2.2 to support changes specified in the ORB Portability Joint Submission (orbos/97-05-15). This submission also addresses issues left open in the current mapping. Elements include:

- Design Rationale
- Mapping of New Types
- Changes to Existing Mapping
- Server-Side Mapping
- Java Stub/Skeleton ORB Interfaces
- Java Class Specification of **org.omg** Classes
- Changes to CORBA 2.2

1.3 Conventions

IDL appears using this font.

Java code appears using this font.

Please note that any change bars have no semantic meaning. They show the places that errata were discovered since the last submission (orbos/98-01-06) was submitted to the OMG. They are present for the convenience of the readers and submitters so that the final edits can be identified.

1.4 Submission Contact Points

All questions about the joint submission should be directed to:

Dan Frantz
BEA Systems, Inc.
436 Amherst St.
Nashua, NH 03063
USA
phone: +1 603 579 2519
fax: +1 603 579 2510
email: dan.frantz@beasys.com

Jeff Mischkinsky
Borland International, Inc.
951 Mariner's Island Blvd. Suite 120
San Mateo, CA 94404
USA
phone: +1 650 312 5158
email: jeffm@visigenic.com

Randy Schnier
International Business Machines, Inc.
Deptartment 143
3605 Highway 52 North
Rochester, MN
USA
phone: +1 507 253 2565
email: rschnier@vnet.ibm.com

Colm Caffrey
IONA Technologies
The IONA Building
8-10 Lower Pembroke Street
Dublin 2, Ireland
phone: +353 1 662 5255
fax: +353 1 662 5244
email: ccaffrey@iona.com

1.5 Proof of Concept

The specification presented here is based on the experience the submitting companies have had in prototyping POA implementations in both Java and C++. The final choices that are embodied in this submission were based upon such prototypes and vendor experience implementing the current mapping.

Response to RFP Requirements 2

The following is a list of the requirements from the Java Language Mapping RFP (OMG orbos/96-08-01), specifying how this submission is responsive to the RFP.

2.1 Map All of IDL

- *The proposed mapping shall map the entire IDL language and provide access to all the feature of the CORBA. In particular the DII/DSI and server side mapping must be covered.*

The entire IDL language, except for the **long double** and **fixed** types, is mapped by the current mapping. This submission extends the current mapping by providing a mapping for the **fixed** type. The **long double** type currently has no equivalent in the Java language, and will remain unmapped at this time.

The Portability submission added a new IDL construct, the **native** type. This submission fully specifies the mapping of the two **native** types specified in the Portability submission.

2.2 Map All the PIDL

- *The mapping of PIDL constructs of the CORBA 2.0 specification shall be described and shall be consistent with the IDL mapping.*

All PIDL types are mapped by the current mapping. This submission provides no new mappings, but does modify some of the previous mappings due to changes specified in the Portability submission.

2.3 Design Rationale

- *The language mapping should be prefaced by an explanation of the design rationale.*

See the chapter on design rationale

2.4 Java Version

- *The version of the Java language specification to be used is 1.0 Beta, unless superseded by a revised specification issued before the submission is due.*

This specification is based on the version of the Java language (and implementation) as defined and released by JavaSoft as part of the JDK 1.1.5 release. However, we have taken great care to guarantee that the mapping is fully compatible with JDK 1.0.2 based environments.

Overall Design Rationale 3

This chapter discusses some of the rationale behind the choices that were made for this mapping.

General issues are discussed in the Introduction. The remaining sections roughly parallel the rest of the submission.

3.1 Introduction

The Portability submission represents a significant change to CORBA. It completely redefines how servers are written in CORBA and in some respects the way clients may interact with those servers. This submission attempts to be true to the spirit of the Portability submission, while trying to maintain as much compatibility as possible with the existing mapping. In some cases, such as the existing server-side mapping, we were forced to deprecate the existing model because of incompatibilities with the Portability submission. In other cases, compatible changes to the mapping were made to support an existing behavior as closely as possible. At all times, the user's point-of-view was considered and much thought was given to usability and performance issues. The final result is a significant step forward in the types of application which can be built with CORBA using the Java language.

3.1.1 Deprecated Methods and Classes

This submission deprecates some CORBA methods and classes that were defined in the previous mapping. Compliant implementations need not provide implementations for deprecated methods or classes. However, all deprecated methods and classes must remain in the **org.omg** package to provide backward compatibility with shipping products implementing the existing mapping. It is the recommendation of this submission that such deprecated methods remain in the **org.omg** package for a period of at least two years to provide vendors and users ample opportunity to update their products to the latest mapping.

All deprecated methods and classes are marked as deprecated using the JDK 1.1 **@deprecated** *javadoc* tag. This mechanism provides users with clear warnings when using deprecated methods in JDK 1.1 or later environments. Each deprecated method and class should indicate the date of deprecation and the OMG document or CORBA version which caused the deprecation.

3.2 Mapping for Extended Types

Since the previous mapping was adopted, JDK 1.1 has been widely adopted throughout the industry and may now be found in several web browsers and integrated development environments. The JDK 1.1 added many new features including a new class which provides an effective mapping of the IDL **fixed** type. This submission maps the IDL **fixed** type to the JDK 1.1 **java.math.BigDecimal** class, with some caveats. The most important restriction being that the mapping is only valid in JDK 1.1 environments or later. The mapping of the **fixed** type is not guaranteed to work in JDK 1.0.2 environments. Great care was taken to avoid references to the **java.math.BigDecimal** class in all standard ORB classes, so there would be no issues in using such classes in a JDK 1.0.2 environment.

No mapping is provided for the IDL **long double** type, since there is still no equivalent in the Java language.

3.3 Mapping for PortableServer::Servant

The mapping for **PortableServer::Servant** was closely modeled after the C++ mapping with a few notable differences. These differences stem from the following requirements:

- Support for non-static Java ORB model
- Support for a standard skeleton model
- Support for Java ease-of-use
- Support for vendor extensions

Unlike other language mappings, the Java mapping has no notion of a global static ORB. Because of this, there must be a way to associate a servant with a particular ORB instance. The Servant mapping specifies the ways in which this may done by both the user and the ORB itself.

The standard skeleton model requires a mechanism by which the ORB may obtain complete type information from a particular Servant. The specified Servant class defines an abstract method to be implemented by skeletons to provide this information.

Ease-of-use was another important factor in the design of the Servant mapping. Unlike C++, where compilers perform no exception checking at compiler time, Java has very strict rules for exception checking. The current mapping states that all user exceptions are checked exceptions in Java, while CORBA system exceptions are unchecked exceptions. The Portability specification defines a large number of new user exceptions, which will complicate the development of code using the POA. To help

alleviate this problem, the Servant class provides methods which provide some common POA functions, but throw CORBA system exceptions instead of the standard POA exceptions.

Although the mapping specifies DSI as the standard skeleton model. The mapping of Servant should not preclude vendors from providing their own skeleton model. The mapping for Servant does not require the use of DSI, nor does the mapping of DSI assume it is the only way to implement servants.

3.3.1 Mapping for Dynamic Skeleton Interface

The mapping for DSI is also based on the C++ mapping, but addresses issues involving the standard skeleton model. Since the standard skeleton model is based on DSI, there are problems with providing a **_this()** method because Java does not allow overloading or overriding of a method's return type. The standard skeleton model also requires hooks for the ORB to obtain type information from the skeleton as discussed above. The ORB requires similar hooks from user DSI code. Rather than define two separate hooks, one mechanism is used to solve both problems.

The **ServerRequest** pseudo object was also updated to match the current C++ definition. Methods defined in the previous definition have been deprecated, and should not be used when writing POA-based DSI code.

3.4 Mapping for Standard Skeletons

The mapping for the standard skeletons was chosen to avoid potential name conflicts with the existing mapping. All generated skeleton classes are prefixed with a **POA_** prefix as in the C++ mapping.

3.5 Support for Optimized In-Process Calls

The POA specifies that all POA operations should behave the same whether an operation is invoked by a local collocated client or over a network via a remote client. While this requirement provides a consistent server-side model, it presents large performance problems for in-process calls where the client and server are collocated.

In the current mapping, in-process calls are very efficient because a client obtains a Java reference to the servant and invokes operations directly, completely bypassing the ORB. This is not possible with the POA, because the POA has the opportunity to redirect the client to a different servant on every invocation.

This submission specifies a new local stub model which provides optimized in-process method invocation, while still providing complete POA semantics. In addition to the new stub model, clarifications were made to the parameter passing modes of IDL interfaces in Java. The new semantics guarantee complete location transparency in compliant programs.

Because the local stub model is for optimization only, ORB vendors are not required to generate local stubs or provide the optimization support in their products. However, ORB vendors must leave all optimization hooks as defined in this submission in place.

3.6 Class Specification of Java org.omg Classes

Since the original IDL to Java mapping was adopted almost a year ago, several vendors have shipped commercial products implementing the specification. Unfortunately, the original mapping was ambiguous or missing definitions of certain classes in the Java **org.omg** package hierarchy. Because vendors often share one another's **org.omg** packages using the mapping's ORB replaceability feature, this presented interoperability problems. Many of these issues have since been resolved through the Java RTF process, but new problems may be introduced as the mapping is changed through further OMG processes.

To avoid this problem in the future, this submission includes a specification for portions of the **org.omg** package and its subpackages which are not defined in standard IDL.

IDL to Java Mapping 4

4.1 Introduction

This section describes changes to the current IDL to Java mapping. The following changes have been made to the mapping:

- Mapping of IDL **fixed** type
- Changes to mapping of interfaces
- Clarification of parameter passing semantics
- Mapping for **native** types

The rationale for design decisions can be found in Chapter 3, “Overall Design Rationale".

4.2 Mapping for IDL Fixed Type

The IDL **fixed** type is mapped to the Java **java.math.BigDecimal** class. Size violations raises a **CORBA::DATA_CONVERSION** exception. Use of fixed types is not supported in Java 1.0 environments.

Because the Java class **java.math.BigDecimal** is not present in Java version 1.0, care must be taken when referring to this class to allow implementors of this specification to create compliant ORBs which run in a Java 1.0 environment. To work around this problem, type safety has been sacrificed for compatibility with Java 1.0. References to fixed types in standard ORB classes use the **java.lang.Number** class, the superclass of **java.math.BigDecimal**, which is defined in Java 1.0. When returning objects in methods whose return signature type is **java.lang.Number**, compliant ORBs must return objects whose true type is **java.math.BigDecimal**. Likewise, callers of ORB functions with method signatures requiring a **java.lang.Number** must pass an object whose true type is **java.math.BigDecimal**. These two rules allow compliant ORBs and user code to simply use casts to convert **java.lang.Number** to **java.math.BigDecimal** without the need to check for the Java **ClassCastException** or verify the type using the

Java **instanceof** operator. Methods which pass or return types of **java.lang.Number** use the name *fixednum* in the method definition as opposed to *fixed*, this is to provide a compatible upgrade path when support for Java 1.0 is no longer deemed important.

4.3 Changes to Mapping for Interfaces

An additional Java interface is now generated for each IDL interface. This class is known as the *operations interface* and is named by appending the suffix **Operations** to the interface name. The operations interface contains all operations specified in the IDL interface definition, and does not extend **org.omg.CORBA.Object**.

The operations interface is used in the server-side mapping and as a mechanism to provide optimized calls for collocated clients and servers.

The Java interface which corresponds to the IDL interface is now defined to extend both the operations interface and **org.omg.CORBA.Object**.

Interface inheritance expressed in IDL is reflected in both the Java interface hierarchy and the Java interface operations hierarchy.

Example

```
// IDL
module Example {
    interface Marker {
    };
    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 BaseOperation {
    void baseOp();
}
public interface ExtendedOperations extends BaseOperations, 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 {
}
public interface Base extends BaseOperations, org.omg.CORBA.Object {
}
public interface Extended extends ExtendedOperations, Base, Marker {
}
```

4.4 Clarification of Parameter Passing Modes

The semantics of parameter passing modes are not defined explicitly enough to support optimized in-process call using the POA. This submission clarifies the semantics of parameter passing modes as follows:

- Java objects passed as IDL **in** parameters are created and owned by the caller. 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. Note: Users may add the Java keyword **final** to each **in** parameter to allow the compiler to assist in enforcing these rules in a JDK 1.1 or later environment.
- 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**.

4.5 Mapping for IDL Native Types

The mapping for IDL native types varies for each actual native type defined in IDL. However, all IDL native types have the following common characteristics:

- A Holder class
- A Helper class

The Holder class for a native type is generated in an identical fashion to Holder classes for other constructed IDL types as defined by section 24.3.1.2 of the CORBA specification. No special consideration is required for native types.

The Helper class for a native type is also generated the same as a Helper class for constructed IDL types as defined by section 24.4 of the CORBA specification, with the following exceptions:

- The **read** method and **write** method definitions must be defined as follows, where **<typename>** indicates the name of the native type:

```
public static <typename> read(org.omg.CORBA.portable.InputStream i) {
    throw new org.omg.CORBA.MARSHAL();
}
public static void write(org.omg.CORBA.portable.OutputStream o, <typename> t) {
    throw new org.omg.CORBA.MARSHAL();
}
```

The definitions of **read** and **write** prevent native types from being marshalled outside of a process address space. Because of this restriction all invocations on interfaces which pass native types as arguments must be invoked using local stubs as defined in Section 6.2, “Local Stubs”, of this document.

4.5.1 TypeCodes for IDL Native Types

All Java Helpers must be able to return a TypeCode which describes the IDL type. This submission introduces TypeCodes for native types to allow for the proper generation of Helper classes for native types.

A new TypeCode kind, **tk_native**, is to be added to **CORBA::TCKind**. The encoding of **tk_native** is the same as the encoding for **tk_objref**. The behavior of the TypeCode pseudo-interface when encapsulating a **tk_native** is similar to that **tk_objref**. The exact changes are specified in Section 9.3, “Changes to TypeCodes”, of this document.

4.5.2 Using Natives with Anys

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

Server-Side Mapping 5

5.1 Introduction

This chapter discusses how object implementations are written in Java. This mapping deprecates all previous standard server-side mappings.

5.2 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. CORBA-compliant ORB implementations are required to provide both of these alternatives. Conforming applications may use either or both of these alternatives.

5.2.1 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 {
    final public org.omg.CORBA.Object _this_object() {...}
    final public org.omg.CORBA.Object _this_object(ORB orb) {...}
    final public ORB _orb() {...}
    final public void _orb(ORB orb) {...}
    final public POA _poa() {...}
    final public byte[] _object_id() {...}
    public POA _default_POA() {...}
    abstract public String[] _all_interfaces(POA poa, byte[] oid);
}
```

The **Servant** class is a Java abstract class which defines six final methods, one method which may be overridden by the user or compiler, and an abstract method which must be supplied by the user or compiler.

The **_orb()** methods provide a method by which the ORB or the user may associate a servant with an instance of an ORB. The **_orb(ORB orb)** method may be called by either the user or the ORB to set the instance of the ORB to be used with a servant. Once it has been set, its value may not be changed, except by first changing it back to **null**. This is to avoid accidental association with multiple ORBs in applications which use multiple ORB instances. Attempting to change the ORB instance to a different value by calling **_orb(ORB orb)** once it has been set results in a **CORBA::BAD_OPERATION** exception. Note, calling **_orb(ORB orb)** mutliple times with the same value is legal, and will not result in an exception being thrown. Attempting to get the value of the ORB using the **_orb()** method will throw a **CORBA::BAD_OPERATION** exception if no instance of an ORB has been associated with the servant.

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.

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()** shall throw 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.

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 is equivalent to invoking **_orb(ORB orb)** followed by the no argument **_this_object()** and is provided as a convenience to users.

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 information for that **oid**. The repository id at the zero index shall represent 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.

5.2.2 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) {
        set_exception(a);
    }
    public void set_exception(Any val) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
}
```

Note, that several methods have been deprecated 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 user must also provide an implementation to the **_all_interfaces()** method declared by the **Servant** class.

5.2.3 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.

5.2.4 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 must 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;

public class POA_<interface_name> 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;

public class POA_A 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) { ... }
}
```

5.2.5 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 POA_<interface_name>_tie extends POA_<interface_name> {
    private <interface_name>Operations _delegate;
    private POA _poa;

    public POA_<interface_name>_tie(<interface_name>Operations delegate) {
        _delegate = delegate;
    }
    public POA_<interface_name>_tie(<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 POA_A_tie extends POA_A {
    private AOperations _delegate;
    private POA _poa;
    public POA_A_tie(AOperations delegate) {
        _delegate = delegate;
    }
    public POA_A_tie(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.

5.3 Mapping for PortableServer::ServantManager

5.3.1 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 is defined as follows:

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

final public class CookieHolder {
    public java.lang.Object value;
    public CookieHolder() {}
    public CookieHolder(java.lang.Object initial) {
        value = initial;
    }
}
```

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.

5.3.2 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.

Java ORB Portability Interfaces 6

6.1 Introduction

The current Java mapping defines a standard stub and skeleton model to provide binary compatibility across multiple ORB vendors. The current mapping provides very high performance for collocated clients and servers, allowing clients to make invocation directly on the server's Java reference. The Portability Submission mandates that the POA has an opportunity to intercept all invocations to a servant including invocations made from within the same address space or Java VM. Using the standard stub model, as specified in CORBA 2.2, would provide no optimization for this situation, and would require clients to marshal all arguments to the ORB, even for a local invocation. This submission specifies a new local stub model designed to address the requirements of the POA, but still provide acceptable performance for calls within the same Java VM.

6.2 Local Stubs

This submission introduces the notion of a local stub. The purpose of a local stub is to provide high performance calls for collocated clients and servers (i.e. clients and servers residing the same Java VM). The local stub API is supported via three additional methods on **org.omg.CORBA.portable.ObjectImpl** and **org.omg.CORBA.portable.Delegate**.

6.2.1 Local Stub APIs

Local stubs are supported through the following new methods and classes:

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

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

abstract public class ObjectImpl implements org.omg.CORBA.Object {
    ...
    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, request);
    }
}

abstract public class Delegate {
    ...
    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 request) {
    }
}
```

The **_is_local()** method is provided so **Helper** classes may determine if a particular object is implemented by a local servant and if so create a local stub, rather than a remote stub. This function would typically be called from within the **narrow()** method of a **Helper** class. The **_is_local()** method may only return true if the servant incarnating the object is located in the same Java VM. The method may return false if the servant is not local or the ORB does not support local stubs for that particular servant. 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 which 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 arguments and returns a **ServantObject** object (Note, ORB vendors may subclass the **ServantObject** object to return 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 may return a null value if it does not wish to support this optimization (e.g. due to security, transactions, etc.). The method must return null 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.

6.2.2 Location Transparency

Local stubs must provide complete location transparency. The client and server behavior must be consistent when using both local and remote stubs. To provide the correct semantics, compliant programs must comply with the new parameter passing semantics defined in Section 4.4, "Clarification of Parameter Passing Modes" of this submission.

6.3 Stub Implementation

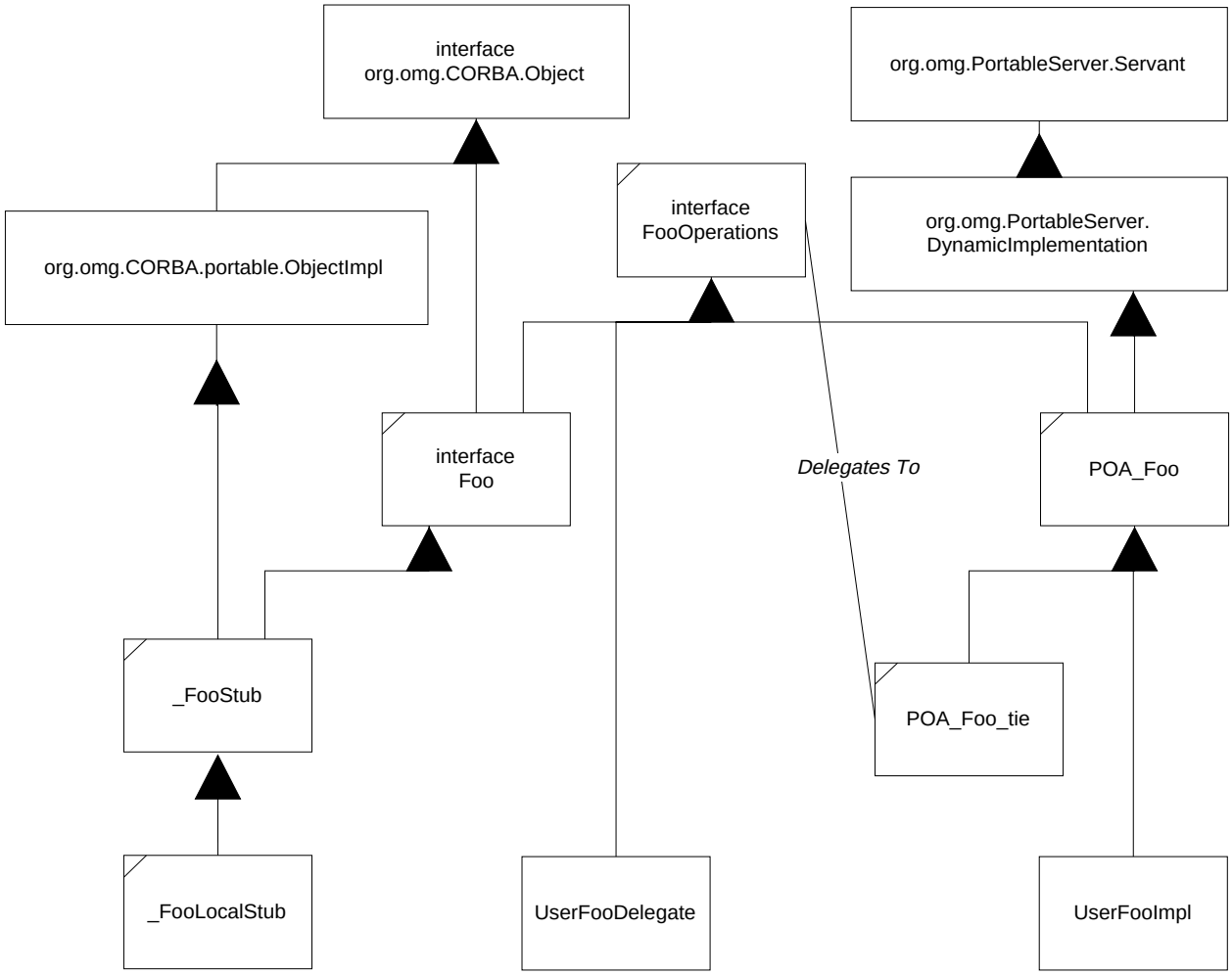
The name of the remote stub is defined to be **_<interface_name>Stub** where **<interface_name>** is the name of the IDL interface. The name of the local stub is defined to be **_<interface_name>LocalStub** where **<interface_name>** is the name of the IDL interface. Local stubs are defined to be direct subclasses of remote stubs.

6.4 Stub and Skeleton Class Hierarchy

An example of the class hierarchy is shown in Figure 6-1. 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 which are generated by the IDL compiler or other tools. Classes beginning with **User** indicate user defined classes which implement interfaces.

Figure 6-1 Class hierarchy for Java portable stubs and skeletons.



Java Class Specification 7

This submission specifies source code for certain Java classes in the **org.omg.CORBA** and **org.omg.PortableServer** packages and subpackages. Compliant implementations must use the specified source code and may not modify any portion of the specified code unless such modifications are explicitly allowed by the specification. Changes to the Java mapping which modify classes in the specified code or add new types which should belong in the specified source code using the rules below, must also modify or provide the specified source code.

7.1 Contents of the Java Class Specification

The Java class specification contains complete Java source code for all portions of the Java mapping which cannot be mapped using the rules for mapping IDL to Java. This includes Java classes for all PIDL, native types, and ORB portability interfaces.

7.2 Allowable Modifications to the Java Class Specification

The following portions of the Java class specification may be modified by ORB vendors:

- 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 **DEFAULT_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 is forbidden.

7.3 Definition of the Java Class Specification

The Java class specification for all classes is provided in Appendix A of this submission.

Conformance Issues 8

8.1 Introduction

This chapter specifies the compliance points for this specification

8.2 Compliance

In order to be conformant with this specification, all the specified mappings and language features must be supported and implemented using the specified semantics.

- There are no optional interfaces
- There are two optional compliance points:
  - Generation of local stubs is not required
  - Implementation of local stub optimization within the ORB is not required
- Note: PIDL specifically excluded by this specification is not required to be mapped.

Changes to CORBA 2.2 9

This section outlines the exact changes which should be made to the existing Java mapping as specified in CORBA 2.2. All changes are shown below.

9.1 Changes to Mapping for Basic Types

9.1.1 Changes to Section 23.3.1 “Introduction”

Add the following to the bottom of Table 23-1, “BASIC TYPE MAPPINGS”.

Figure 23-1 BASIC TYPE MAPPINGS

IDL Type	Java type	Exceptions
fixed	**java.math.BigDecimal**	**CORBA::DATA_CONVERSION**

Replace the text and table in the “Future Support” section with the following paragraph:

> 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 1.1.5, and as a practical matter, they are not yet widely supported by ORB vendors.

Add the following FixedHolder type to the list of primitive holders defined in the “Holder Classes” section.

```
final public class FixedHolder {
    public java.math.BigDecimal value;
    public FixedHolder() {}
    public FixedHolder(java.math.BigDecimal initial) {
        value = initial;
    }
}
```

9.1.1 Changes to Section 23.3.7 “Future Fixed Point Types”

Change title of section 23.3.7 to “Fixed Point Types”. Replace entire contents of section 23.3.7 with Section 4.2, “Mapping for IDL Fixed Type" from this submission.

9.2 Changes to Mapping for Interface

9.2.1 Changes to Section 23.11.1 “Basics”

Replace first two paragraphs with the following paragraphs:

> An IDL **interface** is mapped to two public Java interfaces: a *signature interface* and an *operations interface*. The signature interface 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 interfaces 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. An additional “helper” Java class with the suffix **Helper** appended to the interface name.
>
> The Java operations interface contains the mapped operation signatures. The Java signature interface extends the operations interface and the (mapped) base **org.omg.CORBA.Object**. Methods can be invoked on an object reference to the signature interface. Interface inheritance expressed in IDL is reflected in both the Java signature interface and operations interface hierarchies.

Replace the example with the example from Section 4.3, “Changes to Mapping for Interfaces" from this submission.

9.2.2 Changes to section 23.11.2 “Parameter Passing Modes”

Insert Section 4.4, “Clarification of Parameter Passing Modes" of this submission after the second paragraph of section 23.11.2.

9.3 Changes to TypeCodes

9.3.1 Changes to Section 7.7 “TypeCodes”

Make the following changes to the IDL definition for the TypeCode interface:

- Add new kind **tk_native** to the **CORBA::TCKind** definition.

- Add **tk_native** to the IDL comments above the **id()** and **name()** operations defined in **CORBA::TypeCode**.

Add **tk_native** to Table 7-1, and duplicate the “Parameter List” and “Not Visible” columns from that of **tk_objref**.

Add the following IDL operation to the CORBA::ORB interface defined in Section 7.7.3:

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

9.3.2 Changes to Section 7.8 “Interface Repository”

Propagate changes to TypeCodes from Section 7.7 to IDL defined in this section.

9.3.3 Changes to Section 12.3.4 “Pseudo-Object Types”

Add **tk_native** to Table 12-2, and duplicate the columns “Type” and “Parameter” from that of **tk_objref**. The value of column “Integer Value” must be assigned by the OMG.

9.4 Changes to Mapping for the Any Type

9.4.1 Changes to Section 23.13 “Mapping for the Any Type”

Add the following method definitions to the Java class definition of **Any**:

```
public java.lang.Number extract_fixednum() {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
public void insert_fixednum(java.lang.Number n) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

Add the paragraph from Section 4.5.2, “Using Natives with Anys" of this document as a new paragraph at the end of Section 23.13.

9.5 Changes to ServerRequest and DynamicImplementation

Remove Section 23.16.10, “ServerRequest and Dynamic Implementation”.

9.6 Changes to ORB

Add the following PIDL to the ORB PIDL in Section 23.16.12, “ORB”:

```
// ORB initilization
```

```
boolean work_pending();
void peform_work();
void shutdown(in boolean wait_for_completion);
void run();

// Dynamic Any support

DynAny create_dyn_any(in any value);
DynAny create_basic_dyn_any(in TypeCode type)
        raises(DynAny::InconsistentTypeCode);
DynStruct create_dyn_struct(in TypeCode type)
        raises(DynAny::InconsistentTypeCode);
DynSequence create_dyn_sequence(in TypeCode type)
        raises(DynAny::InconsistentTypeCode);
DynArray create_dyn_array(in TypeCode type)
        raises(DynAny::InconsistentTypeCode);
DynUnion create_dyn_Union(in TypeCode type)
        raises(DynAny::InconsistentTypeCode);
DynEnum create_dyn_enum(in TypeCode type)
        raises(DynAny::InconsistentTypeCode);
DynFixed create_dyn_fixed(in TypeCode type)
        raises(DynAny::InconsistentTypeCode);
```

Add the following Java method definition to the ORB class in Section 23.16.12, “ORB”:

```
    public boolean work_pending() {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public void peform_work() {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    void shutdown(in boolean wait_for_completion) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    void run() {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }

// DynAny support
    public DynAny create_dyn_any(Any value) {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public DynAny create_basic_dyn_any(TypeCode type)
                throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public DynStruct create_dyn_struct(TypeCode type)
                throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public DynSequence create_dyn_sequence(TypeCode type)
                throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public DynArray create_dyn_array(TypeCode type)
                throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public DynUnion create_dyn_union(TypeCode type)
                throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public DynEnum create_dyn_enum(TypeCode type)
                throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
    public DynFixed create_dyn_fixed(TypeCode type)
                throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
        throw new org.omg.CORBA.NO_IMPLEMENT();
    }
```

Add the following javadoc comment to the **connect**, **disconnect** methods and convert methods from abstract to defined throwing a **NO_IMPLEMENT** exception.

```
/**
 * @deprecated Deprecated by Portable Object Adapter, see OMG document
orbos/98-01-06 for details.
 */
```

9.7 Changes to Server-Side Mapping

Replace the entire contents of section 23.7 with the contents of Chapter 5, “Server-Side Mapping" of this submission.

9.8 Changes to Java ORB Portability Interfaces

9.8.1 Changes to Section 23.18.2 “Architecture”

Replace entire contents with the following:

> The stub and skeleton portability architecture allows the use of the DII and DSI as its portability layer. The mapping of the DII and DSI PIDL have operations that support the efficient implementation of portable stubs and skeletons.
>
> All stubs shall 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 mapping defines two types of stubs: remote stubs and local stubs. The remote stub must be named _<interface_name>Stub where <interface_name> is the IDL interface name this stub is implementing. The local stub must be named _<interface_name>LocalStub where <interface_name> is the same IDL interface name. Local stubs are defined to be direct base classes of remote stubs.
>
> All DSI-based skeletons inherit from **org.omg.PortableServer.DynamicImplementation**.

Add new section 23.18.2.1 “Local Stubs” and add the text from Section 6.2, “Local Stubs" of this submission.

Replace the figure in this section with Figure 6-1, “Class hierarchy for Java portable stubs and skeletons.,” on page 34 of this submission.

9.8.2 Changes to Section 23.18.4 “Streaming APIs”

Add the following method definitions to **InputStream**:

```
public java.lang.Number read_fixednum() {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
public voidread_fixednum_array(java.lang.Number[] value,
                               int offset, int length) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

Add the following method definitions to **OutputStream**:

```
public void write_fixednum(java.lang.Number value) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
public void write_fixednum_array(java.lang.Number[], int offset, int length) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
}
```

9.8.3 Changes to Section 23.18.5 "Portability Stub Interfaces"

Add the following new Java class definition:

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

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

Add the following method definitions to the Java ObjectImpl definition:

```
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, request);
}
```

9.8.4 Changes to Section 23.18.6 "Delegate"

Add the following method definitions to the Java Delegate class:

```
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 request) {
}
```

9.9 Java Class Specification

Add Chapter 7, “Java Class Specification" and Appendix A of this submission to the IDL/Java language mapping.

Java Class Specification A

This appendix provides the complete source code specification for certain classes specified as part of this language mapping. The classes specified are as follows:

- org.omg.CORBA
  - Any
  - DynamicImplementation (deprecated)
  - Context
  - ContextList
  - ExceptionList
  - NamedValue
  - NVList
  - Object
  - ORB
  - Principal
  - Request
  - ServerRequest
  - SystemException
  - TypeCode
  - UnknownUserException
  - UserException
- org.omg.CORBA.portable
  - Delegate
  - InputStream
  - ObjectImpl
  - OutputStream
  - ServantObject
  - Streamable

- org.omg.PortableServer
  - DynamicImplementation
  - Servant

A.1 org.omg.CORBA Package

```
package org.omg.CORBA;

abstract public class Any {

  abstract public TypeCode type();
  abstract public void type(TypeCode type);
  abstract public void read_value(org.omg.CORBA.portable.InputStream input,
TypeCode type);
  abstract public void write_value(org.omg.CORBA.portable.OutputStream output);
  abstract public org.omg.CORBA.portable.OutputStream create_output_stream();
  abstract public org.omg.CORBA.portable.InputStream create_input_stream();
  abstract public boolean equal(Any rhs);
  abstract public short extract_short();
  abstract public void insert_short(short value);
  abstract public int extract_long();
  abstract public void insert_long(int value);
  abstract public long extract_longlong();
  abstract public void insert_longlong(long value);
  abstract public short extract_ushort();
  abstract public void insert_ushort(short value);
  abstract public int extract_ulong();
  abstract public void insert_ulong(int value);
  abstract public long extract_ulonglong();
  abstract public void insert_ulonglong(long value);
  abstract public float extract_float();
  abstract public void insert_float(float value);
  abstract public double extract_double();
  abstract public void insert_double(double value);
  abstract public boolean extract_boolean();
  abstract public void insert_boolean(boolean value);
  abstract public char extract_char();
  abstract public void insert_char(char value);
  abstract public char extract_wchar();
  abstract public void insert_wchar(char value);
  abstract public byte extract_octet();
  abstract public void insert_octet(byte value);
  abstract public Any extract_any();
  abstract public void insert_any(Any value);
  abstract public TypeCode extract_TypeCode();
  abstract public void insert_TypeCode(TypeCode value);
  abstract public org.omg.CORBA.Object extract_Object();
  abstract public void insert_Object(org.omg.CORBA.Object value);
  abstract public void insert_Object(org.omg.CORBA.Object value, TypeCode type);
  abstract public String extract_string();
  abstract public void insert_string(String value);
  abstract public String extract_wstring();
  abstract public void insert_wstring(String value);
  abstract public Principal extract_Principal();
```

```
  abstract public void insert_Principal(Principal value);

  abstract public void insert_Streamable(org.omg.CORBA.portable.Streamable
value);
  public Number extract_fixed() {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }
  public void insert_fixed(Number n) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }
}

/**
 * @deprecated Deprecated by the Portable Object Adapter, see OMG document
orbos/98-01-06 for details.
 */
abstract public class DynamicImplementation extends org.omg.CORBA.porta-
ble.ObjectImpl {
  /**
   * @deprecated Deprecated by the Portable Object Adapter, see OMG document
orbos/98-01-06 for details.
   */
  abstract public void invoke(ServerRequest request);
}
abstract public class Context  {
  abstract public java.lang.String context_name();
  abstract public org.omg.CORBA.Context parent();
  abstract public void set_one_value(
    java.lang.String prop_name,
    org.omg.CORBA.Any value
  );
  abstract public void set_values(
    org.omg.CORBA.NVList values
  );
  abstract public org.omg.CORBA.NVList get_values(
    java.lang.String start_scope,
    int op_flags,
    java.lang.String prop_name
  );
  abstract public void delete_values(
    java.lang.String prop_name
  );
  abstract public org.omg.CORBA.Context create_child(
    java.lang.String child_context_name
  );
}
abstract public class ContextList  {
  abstract public int count();
  abstract public void add(
    java.lang.String ctx
  );
```

```
  abstract public java.lang.String item(
    int index
  ) throws
    org.omg.CORBA.Bounds;
  abstract public void remove(
    int index
  ) throws
    org.omg.CORBA.Bounds;
}
abstract public class ExceptionList  {
  abstract public int count();
  abstract public void add(
    org.omg.CORBA.TypeCode exc
  );
  abstract public org.omg.CORBA.TypeCode item(
    int index
  ) throws
    org.omg.CORBA.Bounds;
  abstract public void remove(
    int index
  ) throws
    org.omg.CORBA.Bounds;
}
abstract public class NamedValue  {
  abstract public java.lang.String name();
  abstract public org.omg.CORBA.Any value();
  abstract public int flags();
}
abstract public class NVList  {
  abstract public int count();
  abstract public org.omg.CORBA.NamedValue add(
    int flags
  );
  abstract public org.omg.CORBA.NamedValue add_item(
    java.lang.String name,
    int flags
  );
  abstract public org.omg.CORBA.NamedValue add_value(
    java.lang.String name,
    org.omg.CORBA.Any value,
    int flags
  );
  abstract public org.omg.CORBA.NamedValue item(
    int index
  ) throws
    org.omg.CORBA.Bounds;
  abstract public void remove(
    int index
  ) throws
    org.omg.CORBA.Bounds;
}
```

```
public interface Object {
  public boolean _is_a(
    java.lang.String repId
  );
  public boolean _is_equivalent(
    org.omg.CORBA.Object other_object
  );
  public boolean _non_existent();
  public int _hash(
    int maximum
  );
  public org.omg.CORBA.Object _duplicate();
  public void _release();
  public org.omg.CORBA.InterfaceDef _get_interface();
  public org.omg.CORBA.Request _request(
    java.lang.String operation
  );
  public org.omg.CORBA.Request _create_request(
    org.omg.CORBA.Context ctx,
    java.lang.String operation,
    org.omg.CORBA.NVList arg_list,
    org.omg.CORBA.NamedValue result
  );
  public org.omg.CORBA.Request _create_request(
    org.omg.CORBA.Context ctx,
    java.lang.String operation,
    org.omg.CORBA.NVList arg_list,
    org.omg.CORBA.NamedValue result,
    org.omg.CORBA.ExceptionList exceptions,
    org.omg.CORBA.ContextList contexts
  );
}
abstract public class ORB {

  private static final String DEFAULT_ORB_KEY= "org.omg.CORBA.ORBClass";
  private static final String DEFAULT_ORB_SINGLETON_KEY=
"org.omg.CORBA.ORBSingletonClass";
  private static final String DEFAULT_ORB_VALUE= "";
  private static final String DEFAULT_ORB_SINGLETON_VALUE = "";

  private static ORB _singleton_orb;
  private static Class _thisClass;

  private static String getSystemProperty(String name, String defaultValue) {
    if (_thisClass.getClassLoader() == null) {
      // We were not downloaded in an applet.
      try {
       return System.getProperty(name, defaultValue);
      }
      catch(Exception e) {
```

```
      return defaultValue;
    }
  } else {
    // we must have been loaded by an applet class loader.
    return defaultValue;
  }
}

 private static ORB create(String className) {
  try {
    return (ORB) Class.forName(className).newInstance();
  }
  catch(Exception e) {
    throw new INITIALIZE
     ("Could not instantiate ORB implementation: " + className +
      ": " + e.toString());

  }
}

 static {
  try {
    _thisClass = Class.forName("org.omg.CORBA.ORB");
  } catch (Exception e) {
    throw new INITIALIZE("Could not find class org.omg.CORBA.ORB.class: " +
                e.toString());
  }
   _singleton_orb = create(getSystemProp-
erty(DEFAULT_ORB_SINGLETON_KEY,
                            DEFAULT_ORB_SINGLETON_VALUE));
 }

 public static ORB init() {
  return _singleton_orb;
 }

 public static ORB init(String[] args, java.util.Properties props) {
  String className = null;
  if(props != null) {
    className = props.getProperty(DEFAULT_ORB_KEY);
  }
  if(className == null) {
    className = getSystemProperty(DEFAULT_ORB_KEY,
DEFAULT_ORB_VALUE);
  }
  ORB orb = create(className);
  orb.set_parameters(args, props);
  return orb;
 }

 public static ORB init(java.applet.Applet applet, java.util.Properties props) {
```

```
    String className = applet.getParameter(DEFAULT_ORB_KEY);
    if(className == null && props != null) {
      className = props.getProperty(DEFAULT_ORB_KEY);
    }
    if(className == null) {
      className = DEFAULT_ORB_VALUE;
    }
    ORB orb = create(className);
    orb.set_parameters(applet, props);
    return orb;
  }

  abstract protected void set_parameters(String[] args, java.util.Properties props);

  abstract protected void set_parameters(java.applet.Applet app, java.util.Properties
props);


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

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

  abstract public org.omg.CORBA.NVList create_list(int length);

  abstract public org.omg.CORBA.NVList
create_operation_list(org.omg.CORBA.OperationDef operationDef);

  abstract public org.omg.CORBA.NamedValue create_named_value(String name,
org.omg.CORBA.Any value, int flags);

  abstract public org.omg.CORBA.ExceptionList create_exception_list();

  abstract public org.omg.CORBA.ContextList create_context_list();
  abstract public org.omg.CORBA.Context get_default_context();

  abstract public org.omg.CORBA.Environment create_environment();

  abstract public void send_multiple_requests_oneway(org.omg.CORBA.Request[]
reqs);

  abstract public void send_multiple_requests_deferred(org.omg.CORBA.Request[]
reqs);
  abstract public boolean poll_next_response();

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

  abstract public String[] list_initial_services();
  abstract public org.omg.CORBA.Object resolve_initial_references(String identi-
fier) throws org.omg.CORBA.ORBPackage.InvalidName;
```

```
  abstract public org.omg.CORBA.TypeCode get_primitive_tc(TCKind kind);

  abstract public org.omg.CORBA.TypeCode create_struct_tc(String repository_id,
String type_name, org.omg.CORBA.StructMember[] members);

  abstract public org.omg.CORBA.TypeCode create_union_tc(String repository_id,
String type_name, org.omg.CORBA.TypeCode discriminator_type,
org.omg.CORBA.UnionMember[] members);

  abstract public org.omg.CORBA.TypeCode create_enum_tc(String repository_id,
String type_name, String[] members);

  abstract public org.omg.CORBA.TypeCode create_alias_tc(String repository_id,
String type_name, org.omg.CORBA.TypeCode original_type);

  abstract public org.omg.CORBA.TypeCode create_exception_tc(String
repository_id, String type_name, org.omg.CORBA.StructMember[] members);

  abstract public org.omg.CORBA.TypeCode create_interface_tc(String
repository_id, String type_name);

  abstract public org.omg.CORBA.TypeCode create_string_tc(int length);

  abstract public org.omg.CORBA.TypeCode create_wstring_tc(int length);

  abstract public org.omg.CORBA.TypeCode create_sequence_tc(int length,
org.omg.CORBA.TypeCode element_type);

  abstract public org.omg.CORBA.TypeCode create_recursive_sequence_tc(int
length, int offset);

  abstract public org.omg.CORBA.TypeCode create_array_tc(int length,
org.omg.CORBA.TypeCode element_type);


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

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

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

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

```
  /**
   * @deprecated connect() deprecated, use the Portable Object Adapter (POA)
instead.
   */
  public void connect(org.omg.CORBA.Object object) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

  /**
   * @deprecated disconnect() deprecated, use the Portable Object Adapter (POA)
instead.
   */
  public void disconnect(org.omg.CORBA.Object object) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }


  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 org.omg.CORBA.DynAny create_dyn_any(org.omg.CORBA.Any value) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

  public org.omg.CORBA.DynAny create_basic_dyn_any(org.omg.CORBA.Type-
Code type) throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

  public org.omg.CORBA.DynStruct create_dyn_struct(org.omg.CORBA.TypeCode
type) throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

  public org.omg.CORBA.DynSequence
create_dyn_sequence(org.omg.CORBA.TypeCode type) throws
org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
    throw new org.omg.CORBA.NO_IMPLEMENT();
```

```
  }

  public org.omg.CORBA.DynArray create_dyn_array(org.omg.CORBA.TypeCode
type) throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

  public org.omg.CORBA.DynUnion create_dyn_union(org.omg.CORBA.TypeCode
type) throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

  public org.omg.CORBA.DynEnum create_dyn_enum(org.omg.CORBA.TypeCode
type) throws org.omg.CORBA.DynAnyPackage.InconsistentTypeCode {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

}
abstract public class Principal  {
  abstract public void name(byte[] name);
  abstract public byte[] name();
}
abstract public class Request  {
  abstract public org.omg.CORBA.Object target();
  abstract public java.lang.String operation();
  abstract public org.omg.CORBA.NVList arguments();
  abstract public org.omg.CORBA.NamedValue result();
  abstract public org.omg.CORBA.Environment env();
  abstract public org.omg.CORBA.ExceptionList exceptions();
  abstract public org.omg.CORBA.ContextList contexts();
  abstract public void ctx(org.omg.CORBA.Context ctx);
  abstract public org.omg.CORBA.Context ctx();
  abstract public org.omg.CORBA.Any add_in_arg();
  abstract public org.omg.CORBA.Any add_named_in_arg(
    java.lang.String name
  );
  abstract public org.omg.CORBA.Any add_inout_arg();
  abstract public org.omg.CORBA.Any add_named_inout_arg(
    java.lang.String name
  );
  abstract public org.omg.CORBA.Any add_out_arg();
  abstract public org.omg.CORBA.Any add_named_out_arg(
    java.lang.String name
  );
  abstract public void set_return_type(
    org.omg.CORBA.TypeCode tc
  );
  abstract public org.omg.CORBA.Any return_value();
  abstract public void invoke();
  abstract public void send_oneway();
  abstract public void send_deferred();
```

```
  abstract public void get_response(
  ) throws
    org.omg.CORBA.WrongTransaction;
  abstract public boolean poll_response();
}
abstract public class ServerRequest  {

  /**
   * @deprecated use operation()
   */
  public java.lang.String op_name() {
    return operation();
  }
  public String operation() {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }
  abstract public org.omg.CORBA.Context ctx();

  /**
   * @deprecated use arguments()
   */
  public void params(org.omg.CORBA.NVList params) {
    return arguments();
  }

  public void arguments(org.omg.CORBA.NVList params) {

  }

  /**
   * @deprecated use set_result()
   */
  public void result(org.omg.CORBA.Any result) {
    set_result(result);
  }
  public void set_result(org.omg.CORBA.Any result) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

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

}
abstract public class SystemException extends java.lang.RuntimeException {
```

```
  SystemException(String reason, int minor, CompletionStatus completed) {
    super(reason);
    this.minor = minor;
    this.completed = completed;
  }
  public int minor;

  public CompletionStatus completed;

  public String toString() {
    String result = getClass().getName() + "[";
    if(minor != 0) {
      result += "minor=" + minor + ", ";
    }
    String[] comp = { "YES", "NO", "MAYBE" };
    result += "completed=" + comp[completed.value()];
    if(getMessage() != null) {
      result += ", reason=" + getMessage();
    }
    return result + "]";
  }

}
abstract public class TypeCode  {
  abstract public boolean equal(
    org.omg.CORBA.TypeCode tc
  );
  abstract public org.omg.CORBA.TCKind kind();
  abstract public java.lang.String id(
  ) throws
    org.omg.CORBA.TypeCodePackage.BadKind;
  abstract public java.lang.String name(
  ) throws
    org.omg.CORBA.TypeCodePackage.BadKind;
  abstract public int member_count(
  ) throws
    org.omg.CORBA.TypeCodePackage.BadKind;
  abstract public java.lang.String member_name(
    int index
  ) throws
    org.omg.CORBA.TypeCodePackage.BadKind,
    org.omg.CORBA.TypeCodePackage.Bounds;
  abstract public org.omg.CORBA.TypeCode member_type(
    int index
  ) throws
    org.omg.CORBA.TypeCodePackage.BadKind,
    org.omg.CORBA.TypeCodePackage.Bounds;
  abstract public org.omg.CORBA.Any member_label(
    int index
  ) throws
```

```
    org.omg.CORBA.TypeCodePackage.BadKind,
    org.omg.CORBA.TypeCodePackage.Bounds;
  abstract public org.omg.CORBA.TypeCode discriminator_type(
  ) throws
    org.omg.CORBA.TypeCodePackage.BadKind;
  abstract public int default_index(
  ) throws
    org.omg.CORBA.TypeCodePackage.BadKind;
  abstract public int length(
  ) throws
    org.omg.CORBA.TypeCodePackage.BadKind;
  abstract public org.omg.CORBA.TypeCode content_type(
  ) throws
    org.omg.CORBA.TypeCodePackage.BadKind;
}
final public class UnknownUserException extends org.omg.CORBA.UserException {
  public org.omg.CORBA.Any except;
  public UnknownUserException() {
  }
  public UnknownUserException(
    org.omg.CORBA.Any except
  ) {
    this.except = except;
  }
}
abstract public class UserException extends java.lang.Exception {
}
```

A.2 org.omg.CORBA.portable Package

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

public abstract class Delegate {

  public abstract org.omg.CORBA.ImplementationDef
get_implementation(org.omg.CORBA.Object self);

  public abstract org.omg.CORBA.InterfaceDef
get_interface(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 maximum);

  public abstract org.omg.CORBA.Request request(org.omg.CORBA.Object self,
                               String operation);

  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 exceptions,
                                   org.omg.CORBA.ContextList contexts);

  public abstract org.omg.CORBA.ORB orb(org.omg.CORBA.Object self);

  public boolean is_local(org.omg.CORBA.Object self) {
    return false;
```

```
  }

  public ServantObject servant_preinvoke(org.omg.CORBA.Object self,
                                         String operation) {
    return null;
  }

  public void servant_postinvoke(org.omg.CORBA.Object self,
                                 ServantObject servant) {
  }

}
public abstract class InputStream extends java.io.InputStream {
  public abstract boolean       read_boolean();
  public abstract char          read_char();
  public abstract char          read_wchar();
  public abstract byte          read_octet();
  public abstract short         read_short();
  public abstract short         read_ushort();
  public abstract int           read_long();
  public abstract int           read_ulong();
  public abstract long          read_longlong();
  public abstract long          read_ulonglong();
  public abstract float         read_float();
  public abstract double        read_double();
  public abstract String        read_string();
  public abstract String        read_wstring();
  public abstract org.omg.CORBA.Object read_Object();
  public abstract org.omg.CORBA.TypeCode read_TypeCode();
  public abstract org.omg.CORBA.Any read_any();
  public abstract org.omg.CORBA.Principal read_Principal();
  public abstract void read_boolean_array(boolean[] value, int offset, int length);
  public abstract void read_char_array(char[] value, int offset, int length);
  public abstract void read_wchar_array(char[] value, int offset, int length);
  public abstract void read_octet_array(byte[] value, int offset, int length);
  public abstract void read_short_array(short[] value, int offset, int length);
  public abstract void read_ushort_array(short[] value, int offset, int length);
  public abstract void read_long_array(int[] value, int offset, int length);
  public abstract void read_ulong_array(int[] value, int offset, int length);
  public abstract void read_longlong_array(long[] value, int offset, int length);
  public abstract void read_ulonglong_array(long[] value, int offset, int length);
  public abstract void read_float_array(float[] value, int offset, int length);
  public abstract void read_double_array(double[] value, int offset, int length);

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

  public Number read_fixednum() {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }
```

```
  public void read_fixednum_array(Number[] value, int offset, int length) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

}
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("The delegate has not been
set!");
    }
    return __delegate;
  }

  public void _set_delegate(Delegate delegate) {
    __delegate = delegate;
  }

  abstract public String[] _ids();

  public org.omg.CORBA.ImplementationDef _get_implementation() {
    return _get_delegate().get_implementation(this);
  }

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

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

```
  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.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 org.omg.CORBA.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 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) {
    return _get_delegate().servant_preinvoke(this, operation);
  }

  public void _servant_postinvoke(ServantObject servant) {
    _get_delegate().servant_postinvoke(this, servant);
  }
}
public abstract class OutputStream extends java.io.OutputStream {
  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_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 abstract void write_Principal  (org.omg.CORBA.Principal 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 void write(int b) throws java.io.IOException {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

  public void write_fixednum(Number value) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }

  public void write_fixednum_array(Number[] value, int offset, int length) {
    throw new org.omg.CORBA.NO_IMPLEMENT();
  }
}

public class ServantObject {
  public java.lang.Object servant;
}
public interface Streamable {
  void _read(org.omg.CORBA.portable.InputStream input);
  void _write(org.omg.CORBA.portable.OutputStream output);
  org.omg.CORBA.TypeCode _type();
}
```

A.3 org.omg.PortableServer Package

```
package
abstract public class DynamicImplementation extends Servant {

  abstract public void invoke(org.omg.CORBA.ServerRequest request);

}
import org.omg.CORBA.ORB;
import org.omg.CORBA.ORBPackage.InvalidName;
import org.omg.PortableServer.POAPackage.ServantNotActive;
import org.omg.PortableServer.POAPackage.WrongPolicy;
import org.omg.PortableServer.CurrentPackage.NoContext;

abstract public class Servant {

  private transient ORB _orb = null;
  private transient Current _current = null;

  final public org.omg.CORBA.Object _this_object() {
    POA poa = null;

    try {
      poa = _poa();
    }
    catch (org.omg.CORBA.OBJ_ADAPTER e) {
      // This means we don't have a context for our POACurrent.
      // Use the Servant's default POA instead.
      poa = _default_POA();
    }

    if (poa == null) {
      throw new org.omg.CORBA.OBJ_ADAPTER("null value returned by
_default_POA() on Servant " + this);
    }
    try {
      return poa.servant_to_reference(this);
    }
    catch(ServantNotActive e) {
      throw new org.omg.CORBA.OBJ_ADAPTER(e.toString());
    }
    catch(WrongPolicy e) {
      throw new org.omg.CORBA.OBJ_ADAPTER(e.toString());
    }
  }

  final public org.omg.CORBA.Object _this_object(ORB orb) {
    _orb(orb);
    return _this_object();
  }
```

```
 final public ORB _orb() {
  if (_orb == null) {
   throw new org.omg.CORBA.BAD_OPERATION("The ORB for Servant " + this
+
                                " has not been set.");
  }
  return _orb;
 }

 final public void _orb(ORB orb) {
  if (_orb == null || orb == null) {
   _orb = orb;
   _current = null;
  }
  else if (_orb != orb) {
   throw new org.omg.CORBA.BAD_OPERATION("The ORB for Servant " + this
+
                                " has already been set to " +
                                _orb);
  }
 }

 final public POA _poa() {
  if (_current == null) {
   _getPOACurrent();
  }
  try {
   return _current.get_POA();
  }
  catch(NoContext e) {
   throw new org.omg.CORBA.OBJ_ADAPTER(e.toString());
  }
 }

 final public byte[] _object_id() {
  if (_current == null) {
   _getPOACurrent();
  }
  try {
   return _current.get_object_id();
  }
  catch(NoContext e) {
   throw new org.omg.CORBA.OBJ_ADAPTER(e.toString());
  }
 }

 private synchronized void _getPOACurrent() {
  if (_current == null) {
   try {
    _current =
     CurrentHelper.narrow(_orb().resolve_initial_references("POACurrent"));
```

```
    }
    catch (InvalidName e) {
     throw new org.omg.CORBA.INITIALIZE(e.toString());
    }
   }
  }

  public POA _default_POA() {
   try {
    return POAHelper.narrow(_orb().resolve_initial_references("RootPOA"));
   }
   catch(InvalidName e) {
    throw new org.omg.CORBA.INITIALIZE(e.toString());
   }
  }

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