# IDL-to-Java Generated Files

The files that are generated by the IDL-to-Java compiler, in accordance with the [*IDL-to-Java Language Mapping Specification*](http://cgi.omg.org/cgi-bin/doc?ptc/00-01-08), which is implemented in JavaTM SE 6 according the [compliance](http://docs.google.com/compliance.html) document.

In general IDL names and identifiers are mapped to Java names and identifiers with no change. 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.

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

* The Java class [<type>Helper](#gjdgxs), where <type> is the name of an IDL defined type.
* The Java class [<type>Holder](#30j0zll), 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.
* The Java classes [<interface>Operations](#1fob9te), <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.   
  Helper Files  
  Helper files supply several static methods needed to manipulate the type. 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 include narrow operation(s). The static narrow method 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.  
    
  Holder Files  
  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.)  
  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.   
  Operations Files  
  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.  
  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.   
  Stubs  
  For the mapping of a non-object-oriented language, there will be a programming interface to the stubs for each interface type. Generally, the stubs will present access to the OMG IDL-defined operations on an object in a way that is easy for programmers to predict once they are familiar with OMG IDL and the language mapping for the particular programming language. The stubs make calls on the rest of the ORB using interfaces that are private to, and presumably optimized for, the particular ORB Core. If more than one ORB is available, there may be different stubs corresponding to the different ORBs. In this case, it is necessary for the ORB and language mapping to cooperate to associate the correct stubs with the particular object reference.  
  Object-oriented programming languages, such as Java, C++, and Smalltalk, do not require stub interfaces.