java.io

Interface Serializable

All Known Subinterfaces:

AdapterActivator, Attribute, Attribute, Attributes, BindingIterator, CertPathValidatorException.Reason, ClientRequestInfo, ClientRequestInterceptor, Codec, CodecFactory, Control, Current, Current, Current, CustomValue, DataInputStream, DataOutputStream, Descriptor, DHPrivateKey, DHPublicKey, DocAttribute, DomainManager, DSAPrivateKey, DSAPublicKey, DynAny, DynAnyFactory, DynArray, DynEnum, DynFixed, DynSequence, DynStruct, DynUnion, DynValue, DynValueBox, DynValueCommon, ECPrivateKey, ECPublicKey, ExtendedRequest, ExtendedResponse, Externalizable, IdAssignmentPolicy, IDLEntity, IDLType, IdUniquenessPolicy, ImplicitActivationPolicy, Interceptor, IORInfo, IORInterceptor, IORInterceptor_3_0, IRObject, Key, LifespanPolicy, Name, NamingContext, NamingContextExt, NotificationFilter, ObjectReferenceFactory, ObjectReferenceTemplate, ORBInitializer, ORBInitInfo, PBEKey, POA, POAManager, Policy, PolicyFactory, PrintJobAttribute, PrintRequestAttribute, PrintServiceAttribute, PrivateKey, PublicKey, QueryExp, RelationType, RemoteRef, RequestInfo, RequestProcessingPolicy, RSAMultiPrimePrivateCrtKey, RSAPrivateCrtKey, RSAPrivateKey, RSAPublicKey, RunTime, SecretKey, ServantActivator, ServantLocator, ServantManager, ServantRetentionPolicy, ServerRef, ServerRequestInfo, ServerRequestInterceptor, StreamableValue, SupportedValuesAttribute, ThreadPolicy, UnsolicitedNotification, ValueBase, ValueExp

---

`public interface` **`Serializable`**

Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable.

To allow subtypes of non-serializable classes to be serialized, the subtype may assume responsibility for saving and restoring the state of the supertype's public, protected, and (if accessible) package fields. The subtype may assume this responsibility only if the class it extends has an accessible no-arg constructor to initialize the class's state. It is an error to declare a class Serializable if this is not the case. The error will be detected at runtime.

During deserialization, the fields of non-serializable classes will be initialized using the public or protected no-arg constructor of the class. A no-arg constructor must be accessible to the subclass that is serializable. The fields of serializable subclasses will be restored from the stream.

When traversing a graph, an object may be encountered that does not support the Serializable interface. In this case the NotSerializableException will be thrown and will identify the class of the non-serializable object.

Classes that require special handling during the serialization and deserialization process must implement special methods with these exact signatures:

```
 private void writeObject(java.io.ObjectOutputStream out)
     throws IOException
 private void readObject(java.io.ObjectInputStream in)
     throws IOException, ClassNotFoundException;
 private void readObjectNoData()
     throws ObjectStreamException;
```

The writeObject method is responsible for writing the state of the object for its particular class so that the corresponding readObject method can restore it. The default mechanism for saving the Object's fields can be invoked by calling out.defaultWriteObject. The method does not need to concern itself with the state belonging to its superclasses or subclasses. State is saved by writing the individual fields to the ObjectOutputStream using the writeObject method or by using the methods for primitive data types supported by DataOutput.

The readObject method is responsible for reading from the stream and restoring the classes fields. It may call in.defaultReadObject to invoke the default mechanism for restoring the object's non-static and non-transient fields. The defaultReadObject method uses information in the stream to assign the fields of the object saved in the stream with the correspondingly named fields in the current object. This handles the case when the class has evolved to add new fields. The method does not need to concern itself with the state belonging to its superclasses or subclasses. State is saved by writing the individual fields to the ObjectOutputStream using the writeObject method or by using the methods for primitive data types supported by DataOutput.

The readObjectNoData method is responsible for initializing the state of the object for its particular class in the event that the serialization stream does not list the given class as a superclass of the object being deserialized. This may occur in cases where the receiving party uses a different version of the deserialized instance's class than the sending party, and the receiver's version extends classes that are not extended by the sender's version. This may also occur if the serialization stream has been tampered; hence, readObjectNoData is useful for initializing deserialized objects properly despite a "hostile" or incomplete source stream.

Serializable classes that need to designate an alternative object to be used when writing an object to the stream should implement this special method with the exact signature:

```
 ANY-ACCESS-MODIFIER Object writeReplace() throws ObjectStreamException;
```

This writeReplace method is invoked by serialization if the method exists and it would be accessible from a method defined within the class of the object being serialized. Thus, the method can have private, protected and package-private access. Subclass access to this method follows java accessibility rules.

Classes that need to designate a replacement when an instance of it is read from the stream should implement this special method with the exact signature.

```
 ANY-ACCESS-MODIFIER Object readResolve() throws ObjectStreamException;
```

This readResolve method follows the same invocation rules and accessibility rules as writeReplace.

The serialization runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization. If the receiver has loaded a class for the object that has a different serialVersionUID than that of the corresponding sender's class, then deserialization will result in an `InvalidClassException`. A serializable class can declare its own serialVersionUID explicitly by declaring a field named `"serialVersionUID"` that must be static, final, and of type `long`:

```
 ANY-ACCESS-MODIFIER static final long serialVersionUID = 42L;
```

If a serializable class does not explicitly declare a serialVersionUID, then the serialization runtime will calculate a default serialVersionUID value for that class based on various aspects of the class, as described in the Java(TM) Object Serialization Specification. However, it is *strongly recommended* that all serializable classes explicitly declare serialVersionUID values, since the default serialVersionUID computation is highly sensitive to class details that may vary depending on compiler implementations, and can thus result in unexpected `InvalidClassExceptions` during deserialization. Therefore, to guarantee a consistent serialVersionUID value across different java compiler implementations, a serializable class must declare an explicit serialVersionUID value. It is also strongly advised that explicit serialVersionUID declarations use the `private` modifier where possible, since such declarations apply only to the immediately declaring class--serialVersionUID fields are not useful as inherited members. Array classes cannot declare an explicit serialVersionUID, so they always have the default computed value, but the requirement for matching serialVersionUID values is waived for array classes.

Since:

JDK1.1

See Also:

`ObjectOutputStream`, `ObjectInputStream`, `ObjectOutput`, `ObjectInput`, `Externalizable`