Université Joseph Fourier
MIAGE
Année 2007-2008

UFR - IMAG
Licence 3

La sérialisation

Sébastien Laborie et Philippe Morat

1 Objectifs

Construire des classes dites sérialisables. Sérialiser et désérialiser des objets.

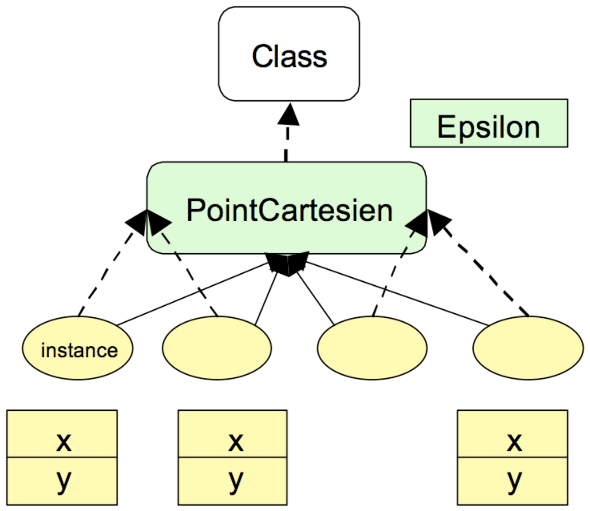
2 Rappel

Le concept de “sérialisation" permet de lire ou d'écrire l'état des objets. Il ne s'agit pas, lorsqu'il s'agit d'écrire un objet, de donner une représentation textuelle d'un objet, mais de donner une représentation binaire. Cette représentation bien que propriétaire sera évidemment indépendante de la plate-forme utilisée donc portable.

En Java, les classes *ObjectInputStream* et *ObjectOutputStream*, dont les documentations sont fournies en annexe, sont utilisées pour pouvoir sérialiser des objets.

3 Sérialiser des points cartésiens

Soit la classe `PointCartesien` vue dans les précédents TD et représentée ci-dessous :



On désire utiliser une sérialisation standard pour sauvegarder des instances de *PointCartesien*.

Question 1 *Que faut-il spécifier dans la classe* `PointCartesien` *pour pouvoir sérialiser ce type d'objets ?*

On souhaite maintenant sérialiser chaque point cartésien contenu dans une liste. Pour cela, on spécifie la méthode suivante : `public void sauver(Vector liste){...}`.

Question 2 *Ecrire le contenu de la méthode* sauver *à l'aide de la classe* ObjectOutputStream.

Question 3 *Existe-t-il une technique plus simple évitant de sérialiser manuellement chaque point cartésien contenu dans une liste ?*

4 Désérialiser des points cartésiens

La désérialisation consiste à restituer l'état d'un objet ayant été sérialisé. Lors de la question précédente nous avons sérialiser un ensemble de points cartésiens. Nous souhaitons dans cette partie désérialiser cet ensemble de points.

Pour cela, on spécifie la méthode suivante : `public Vector charger(InputStream input){...}`.

Question 4 *Ecrire le contenu de la méthode* charger *à l'aide de la classe* ObjectInputStream.

Question 5 *Est-il possible dans l'état actuel de récupérer les données contenues dans la variable static* epsilon *?*

5 Contrôler la sérialisation

Question 6 *Que faut-il modifier dans les méthodes* sauver *et* charger *pour sérialiser la variable static* epsilon *?*

On souhaite ne sérialiser que les abscisses des points cartésiens.

Question 7 *En modifiant la classe* PointCartesien*, donner deux solutions pour ne sérialiser que les abcisses des objets* PointCartesien.

java.io

Class ObjectInputStream

```
java.lang.Object
  └ java.io.InputStream
      └ java.io.ObjectInputStream
```

All Implemented Interfaces:

DataInput, ObjectInput, ObjectStreamConstants

---

public class **ObjectInputStream**
extends InputStream
implements ObjectInput, ObjectStreamConstants

An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.

ObjectOutputStream and ObjectInputStream can provide an application with persistent storage for graphs of objects when used with a FileOutputStream and FileInputStream respectively. ObjectInputStream is used to recover those objects previously serialized. Other uses include passing objects between hosts using a socket stream or for marshaling and unmarshaling arguments and parameters in a remote communication system.

ObjectInputStream ensures that the types of all objects in the graph created from the stream match the classes present in the Java Virtual Machine. Classes are loaded as required using the standard mechanisms.

Only objects that support the java.io.Serializable or java.io.Externalizable interface can be read from streams.

The method `readObject` is used to read an object from the stream. Java's safe casting should be used to get the desired type. In Java, strings and arrays are objects and are treated as objects during serialization. When read they need to be cast to the expected type.

Primitive data types can be read from the stream using the appropriate method on DataInput.

The default deserialization mechanism for objects restores the contents of each field to the value and type it had when it was written. Fields declared as transient or static are ignored by the deserialization process. References to other objects cause those objects to be read from the stream as necessary. Graphs of objects are restored correctly using a reference sharing mechanism. New objects are always allocated when deserializing, which prevents existing objects from being overwritten.

Reading an object is analogous to running the constructors of a new object. Memory is allocated for the object and initialized to zero (NULL). No-arg constructors are invoked for the non-serializable classes and then the fields of the serializable classes are restored from the stream starting with the serializable class closest to java.lang.object and finishing with the object's most specific class.

For example to read from a stream as written by the example in ObjectOutputStream:

```
        FileInputStream fis = new FileInputStream("t.tmp");
        ObjectInputStream ois = new ObjectInputStream(fis);

        int i = ois.readInt();
        String today = (String) ois.readObject();
        Date date = (Date) ois.readObject();

        ois.close();
```

Classes control how they are serialized by implementing either the java.io.Serializable or java.io.Externalizable interfaces.

Implementing the Serializable interface allows object serialization to save and restore the entire state of the object and it allows classes to evolve between the time the stream is written and the time it is read. It automatically traverses references between objects, saving and restoring entire graphs.

Serializable classes that require special handling during the serialization and deserialization process should implement the following methods:

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

The readObject method is responsible for reading and restoring the state of the object for its particular class using data written to the stream by the corresponding writeObject method. The method does not need to concern itself with the state belonging to its superclasses or subclasses. State is restored by reading data from the ObjectInputStream for the individual fields and making assignments to the appropriate fields of the object. Reading primitive data types is supported by DataInput.

Any attempt to read object data which exceeds the boundaries of the custom data written by the corresponding writeObject method will cause an OptionalDataException to be thrown with an eof field value of true. Non-object reads which exceed the end of the allotted data will reflect the end of data in the same way that they would indicate the end of the stream: bytewise reads will return -1 as the byte read or number of bytes read, and primitive reads will throw EOFExceptions. If there is no corresponding writeObject method, then the end of default serialized data marks the end of the allotted data.

Primitive and object read calls issued from within a readExternal method behave in the same manner--if the stream is already positioned at the end of data written by the corresponding writeExternal method, object reads will throw OptionalDataExceptions with eof set to true, bytewise reads will return -1, and primitive reads will throw EOFExceptions. Note that this behavior does not hold for streams written with the old `ObjectStreamConstants.PROTOCOL_VERSION_1` protocol, in which the end of data written by writeExternal methods is not demarcated, and hence cannot be detected.

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.

Serialization does not read or assign values to the fields of any object that does not implement the java.io.Serializable interface. Subclasses of Objects that are not serializable can be serializable. In this case the non-serializable class must have a no-arg constructor to allow its fields to be initialized. In this case it is the responsibility of the subclass to save and restore the state of the non-serializable class. It is frequently the case that the fields of that class are accessible (public, package, or protected) or that there are get and set methods that can be used to restore the state.

Any exception that occurs while deserializing an object will be caught by the ObjectInputStream and abort the reading process.

Implementing the Externalizable interface allows the object to assume complete control over the contents and format of the object's serialized form. The methods of the Externalizable interface, writeExternal and readExternal, are called to save and restore the objects state. When implemented by a class they can write and read their own state using all of the methods of ObjectOutput and ObjectInput. It is the responsibility of the objects to handle any versioning that occurs.

Since:
JDK1.1

See Also:
`DataInput`, `ObjectOutputStream`, `Serializable`, Object Serialization Specification, Section 3, Object Input Classes

---

Nested Class Summary

`static class`	**`ObjectInputStream.GetField`** Provide access to the persistent fields read from the input stream.

Field Summary

Fields inherited from interface java.io.ObjectStreamConstants
`baseWireHandle`, `PROTOCOL_VERSION_1`, `PROTOCOL_VERSION_2`, `SC_BLOCK_DATA`, `SC_EXTERNALIZABLE`, `SC_SERIALIZABLE`, `SC_WRITE_METHOD`, `STREAM_MAGIC`, `STREAM_VERSION`, `SUBCLASS_IMPLEMENTATION_PERMISSION`, `SUBSTITUTION_PERMISSION`, `TC_ARRAY`, `TC_BASE`, `TC_BLOCKDATA`, `TC_BLOCKDATALONG`, `TC_CLASS`, `TC_CLASSDESC`, `TC_ENDBLOCKDATA`, `TC_EXCEPTION`, `TC_LONGSTRING`, `TC_MAX`, `TC_NULL`, `TC_OBJECT`, `TC_PROXYCLASSDESC`, `TC_REFERENCE`, `TC_RESET`, `TC_STRING`

Constructor Summary

`protected`	**`ObjectInputStream`**`()` Provide a way for subclasses that are completely reimplementing ObjectInputStream to not have to allocate private data just used by this implementation of ObjectInputStream.
	`ObjectInputStream``(InputStream in)` Creates an ObjectInputStream that reads from the specified InputStream.

Method Summary

`int`	**`available`**`()` Returns the number of bytes that can be read without blocking.
`void`	**`close`**`()` Closes the input stream.
`void`	**`defaultReadObject`**`()` Read the non-static and non-transient fields of the current class from this stream.
`protected boolean`	**`enableResolveObject`**`(boolean enable)` Enable the stream to allow objects read from the stream to be replaced.

`int`	**read**`()` Reads a byte of data.
`int`	**read**`(byte[] buf, int off, int len)` Reads into an array of bytes.
`boolean`	**readBoolean**`()` Reads in a boolean.
`byte`	**readByte**`()` Reads an 8 bit byte.
`char`	**readChar**`()` Reads a 16 bit char.
`protected ObjectStreamClass`	**readClassDescriptor**`()` Read a class descriptor from the serialization stream.
`double`	**readDouble**`()` Reads a 64 bit double.
`ObjectInputStream.GetField`	**readFields**`()` Reads the persistent fields from the stream and makes them available by name.
`float`	**readFloat**`()` Reads a 32 bit float.
`void`	**readFully**`(byte[] buf)` Reads bytes, blocking until all bytes are read.
`void`	**readFully**`(byte[] buf, int off, int len)` Reads bytes, blocking until all bytes are read.
`int`	**readInt**`()` Reads a 32 bit int.
`String`	**readLine**`()` **Deprecated.** *This method does not properly convert bytes to characters. see DataInputStream for the details and alternatives.*
`long`	**readLong**`()` Reads a 64 bit long.
`Object`	**readObject**`()` Read an object from the ObjectInputStream.
`protected Object`	**readObjectOverride**`()` This method is called by trusted subclasses of ObjectOutputStream that constructed ObjectOutputStream using the protected no-arg constructor.
`short`	**readShort**`()` Reads a 16 bit short.
`protected void`	**readStreamHeader**`()` The readStreamHeader method is provided to allow subclasses to read and verify their own stream headers.
`Object`	**readUnshared**`()` Reads an "unshared" object from the ObjectInputStream.
`int`	**readUnsignedByte**`()` Reads an unsigned 8 bit byte.
`int`	**readUnsignedShort**`()` Reads an unsigned 16 bit short.

`String`	`readUTF()` Reads a UTF format String.
`void`	`registerValidation(ObjectInputValidation obj, int prio)` Register an object to be validated before the graph is returned.
`protected Class`	`resolveClass(ObjectStreamClass desc)` Load the local class equivalent of the specified stream class description.
`protected Object`	`resolveObject(Object obj)` This method will allow trusted subclasses of ObjectInputStream to substitute one object for another during deserialization.
`protected Class`	`resolveProxyClass(String[] interfaces)` Returns a proxy class that implements the interfaces named in a proxy class descriptor; subclasses may implement this method to read custom data from the stream along with the descriptors for dynamic proxy classes, allowing them to use an alternate loading mechanism for the interfaces and the proxy class.
`int`	`skipBytes(int len)` Skips bytes, block until all bytes are skipped.

Methods inherited from class java.io.InputStream
`mark, markSupported, read, reset, skip`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from interface java.io.ObjectInput
`read, skip`

Submit a bug or feature
For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

java.io

Class ObjectOutputStream

```
java.lang.Object
  └ java.io.OutputStream
      └ java.io.ObjectOutputStream
```

All Implemented Interfaces:

DataOutput, ObjectOutput, ObjectStreamConstants

public class **ObjectOutputStream**
extends OutputStream
implements ObjectOutput, ObjectStreamConstants

An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream. The objects can be read (reconstituted) using an ObjectInputStream. Persistent storage of objects can be accomplished by using a file for the stream. If the stream is a network socket stream, the objects can be reconsituted on another host or in another process.

Only objects that support the java.io.Serializable interface can be written to streams. The class of each serializable object is encoded including the class name and signature of the class, the values of the object's fields and arrays, and the closure of any other objects referenced from the initial objects.

The method writeObject is used to write an object to the stream. Any object, including Strings and arrays, is written with writeObject. Multiple objects or primitives can be written to the stream. The objects must be read back from the corresponding ObjectInputstream with the same types and in the same order as they were written.

Primitive data types can also be written to the stream using the appropriate methods from DataOutput. Strings can also be written using the writeUTF method.

The default serialization mechanism for an object writes the class of the object, the class signature, and the values of all non-transient and non-static fields. References to other objects (except in transient or static fields) cause those objects to be written also. Multiple references to a single object are encoded using a reference sharing mechanism so that graphs of objects can be restored to the same shape as when the original was written.

For example to write an object that can be read by the example in ObjectInputStream:

```
        FileOutputStream fos = new FileOutputStream("t.tmp");
        ObjectOutputStream oos = new ObjectOutputStream(fos);

        oos.writeInt(12345);
        oos.writeObject("Today");
        oos.writeObject(new Date());

        oos.close();
```

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

```
private void readObject(java.io.ObjectInputStream stream)
    throws IOException, ClassNotFoundException;
private void writeObject(java.io.ObjectOutputStream stream)
    throws IOException
```

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 method does not need to concern itself with the state belonging to the object's 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.

Serialization does not write out the fields of any object that does not implement the java.io.Serializable interface. Subclasses of Objects that are not serializable can be serializable. In this case the non-serializable class must have a no-arg constructor to allow its fields to be initialized. In this case it is the responsibility of the subclass to save and restore the state of the non-serializable class. It is frequently the case that the fields of that class are accessible (public, package, or protected) or that there are get and set methods that can be used to restore the state.

Serialization of an object can be prevented by implementing writeObject and readObject methods that throw the NotSerializableException. The exception will be caught by the ObjectOutputStream and abort the serialization process.

Implementing the Externalizable interface allows the object to assume complete control over the contents and format of the object's serialized form. The methods of the Externalizable interface, writeExternal and readExternal, are called to save and restore the objects state. When implemented by a class they can write and read their own state using all of the methods of ObjectOutput and ObjectInput. It is the responsibility of the objects to handle any versioning that occurs.

Primitive data, excluding serializable fields and externalizable data, is written to the ObjectOutputStream in block-data records. A block data record is composed of a header and data. The block data header consists of a marker and the number of bytes to follow the header. Consecutive primitive data writes are merged into one block-data record. The blocking factor used for a block-data record will be 1024 bytes. Each block-data record will be filled up to 1024 bytes, or be written whenever there is a termination of block-data mode. Calls to the ObjectOutputStream methods writeObject, defaultWriteObject and writeFields initially terminate any existing block-data record.

Since:
    JDK1.1
See Also:
    `DataOutput`, `ObjectInputStream`, `Serializable`, `Externalizable`, Object Serialization Specification, Section 2, Object Output Classes

Nested Class Summary

`static class`	**`ObjectOutputStream.PutField`** Provide programatic access to the persistent fields to be written to ObjectOutput.

Field Summary

Fields inherited from interface java.io.ObjectStreamConstants

`baseWireHandle`, `PROTOCOL_VERSION_1`, `PROTOCOL_VERSION_2`, `SC_BLOCK_DATA`, `SC_EXTERNALIZABLE`, `SC_SERIALIZABLE`, `SC_WRITE_METHOD`, `STREAM_MAGIC`, `STREAM_VERSION`, `SUBCLASS_IMPLEMENTATION_PERMISSION`, `SUBSTITUTION_PERMISSION`, `TC_ARRAY`, `TC_BASE`, `TC_BLOCKDATA`, `TC_BLOCKDATALONG`, `TC_CLASS`, `TC_CLASSDESC`, `TC_ENDBLOCKDATA`, `TC_EXCEPTION`, `TC_LONGSTRING`, TC_MAX, TC_NULL, TC_OBJECT, TC_PROXYCLASSDESC, TC_REFERENCE, TC_RESET, TC_STRING

Constructor Summary

protected	**ObjectOutputStream**() Provide a way for subclasses that are completely reimplementing ObjectOutputStream to not have to allocate private data just used by this implementation of ObjectOutputStream.
	ObjectOutputStream(OutputStream out) Creates an ObjectOutputStream that writes to the specified OutputStream.

Method Summary

protected void	**annotateClass**(Class cl) Subclasses may implement this method to allow class data to be stored in the stream.
protected void	**annotateProxyClass**(Class cl) Subclasses may implement this method to store custom data in the stream along with descriptors for dynamic proxy classes.
void	**close**() Closes the stream.
void	**defaultWriteObject**() Write the non-static and non-transient fields of the current class to this stream.
protected void	**drain**() Drain any buffered data in ObjectOutputStream.
protected boolean	**enableReplaceObject**(boolean enable) Enable the stream to do replacement of objects in the stream.
void	**flush**() Flushes the stream.
ObjectOutputStream.PutField	**putFields**() Retrieve the object used to buffer persistent fields to be written to the stream.
protected Object	**replaceObject**(Object obj) This method will allow trusted subclasses of ObjectOutputStream to substitute one object for another during serialization.
void	**reset**() Reset will disregard the state of any objects already written to the stream.
void	**useProtocolVersion**(int version) Specify stream protocol version to use when writing the stream.
void	**write**(byte[] buf) Writes an array of bytes.
void	**write**(byte[] buf, int off, int len) Writes a sub array of bytes.
void	**write**(int val) Writes a byte.

`void`	`writeBoolean(boolean val)` Writes a boolean.
`void`	`writeByte(int val)` Writes an 8 bit byte.
`void`	`writeBytes(String str)` Writes a String as a sequence of bytes.
`void`	`writeChar(int val)` Writes a 16 bit char.
`void`	`writeChars(String str)` Writes a String as a sequence of chars.
`protected void`	`writeClassDescriptor(ObjectStreamClass desc)` Write the specified class descriptor to the ObjectOutputStream.
`void`	`writeDouble(double val)` Writes a 64 bit double.
`void`	`writeFields()` Write the buffered fields to the stream.
`void`	`writeFloat(float val)` Writes a 32 bit float.
`void`	`writeInt(int val)` Writes a 32 bit int.
`void`	`writeLong(long val)` Writes a 64 bit long.
`void`	`writeObject(Object obj)` Write the specified object to the ObjectOutputStream.
`protected void`	`writeObjectOverride(Object obj)` Method used by subclasses to override the default writeObject method.
`void`	`writeShort(int val)` Writes a 16 bit short.
`protected void`	`writeStreamHeader()` The writeStreamHeader method is provided so subclasses can append or prepend their own header to the stream.
`void`	`writeUnshared(Object obj)` Writes an "unshared" object to the ObjectOutputStream.
`void`	`writeUTF(String str)` Primitive data write of this String in UTF format.

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`