**2.4 Serialization**

Serialization enables the state of a Java program to be captured and written out to a byte stream [Sun 04b]. This allows for the state to be preserved so that it can be reinstated (by deserialization). Serialization also allows for Java method calls to be transmitted over a network for Remote Method Invocation (RMI). An object (called someObject below) can be serialized as follows:

An object (called someObject below) can be serialized as follows:

ObjectOutputStream oos = new ObjectOutputStream ( new FileOutputStream (“SerialOutput”) );

oos.writeObject (someObject);

oos.flush ( );

The object can be deserialized as follows:

ObjectInputStream ois = new ObjectInputStream ( new FileInputStream (“SerialOutput”) );

someObject = (SomeClass)ois.readObject ( );

Serialization captures all the fields of a class, provided the class implements the Serializable interface, including the non-public fields that are not normally accessible (unless the field is declared transient). If the byte stream to which the serialized values are written is readable, then the values of the normally inaccessible fields may be read. Moreover, it may be possible to modify or forge the preserved values so that when the class is deserialized, the values become corrupted.

Introducing a security manager does not prevent the normally inaccessible fields from being serialized and deserialized (although permission must be granted to write to and read from the file or network if the byte stream is being stored or transmitted). Network traffic (including RMI) can be protected, however, by using SSL.

