Serialization

Serialization of Objects

Goal: transform a network of objects into a "serial form", i.e., a sequence of bytes

- also called marshalling

Application:

- store in a file
- transmit over a network

Inverse operation: deserialization

- In deserialization we reconstruct the objects from a sequence of bytes

Serialization in Java

In the packet java.io we have the class ObjectOutputStream, which enables us to serialize objects

The class ObjectOutputStream contains different methods to serialize the corresponding primitive data types, e.g.:

- public void writeInt(int data) throws IOException
- ...

However, the most important method of the class ObjectOutputStream is

- public void writeObject(Object o) throws IOException

which allows to serialize an object

Serialization in Java (2)

The method writeObject is writing the following data into its OutputStream

- Information about the class of the object that is passed to the function
- All non-static and non-transient attributes of the object that is passed to the function. This includes attributes that have been inherited from a super-class

Static attributes will not be serialized as they do not belong to the object but to the class of the object

Transient Attributes

Attributes, that should not be serialized can be marked as transient:

```
class MyClass {
   transient int temporary_attribute;
   ...
}
```

Typical examples for such attributes are attributes whose values are only relevant in certain stages of a program or that are used to communicate data between different operations

Serializable Classes

Classes, whose objects we want to serialize, have to implement the interface Serializable

```
public interface Serializable {
        // no methods!
}
```

Serializable is a so called marker-interface:

- marks the objects of the class as serializable

Serializable Classes

The interface Serializable does not define methods but at runtime it is checked when calling writeObject whether the object to be serialized implements this interface

Otherwise:

- NotSerializableException

Serializability will be inherited

Almost all standard classes of Java are Serializable with a few exceptions:

- open files or network connections are not Serializable
- if you have such attributes in your own classes they should be marked as transient

Example

```
import java.io.*;
public class WriteObjects{
    public static void main(String[] args){
      try {
            FileOutputStream fs = new FileOutputStream("test.ser");
            ObjectOutputStream os = new ObjectOutputStream(fs);
            os.writeInt(123);
            os.writeObject("Hallo");
            os.close();
      } catch (IOException e) { System.err.println(e.toString()); }
    }
}
```

Deserialization

To reconstruct data from a serialized format we have the class ObjectInputStream from the packet java.io

Analogously to ObjectOutputStream we have methods to deserialize primitive data types, e.g.

- public int readInt() throws IOException
- ...

However, the most important method is

- public Object readObject() throws ClassNotFoundException, IOException

that allows to reconstruct a serialized Object

Distributed Systems

Deserialization

Deserialization of an object consists of the following basic steps:

- a new object of the class to be deserialized is created
- attributes are initialized with standard values
- the default constructor of the first non-serializable super-class is called
- finally, all serializable data is read from the stream and used to assign the values to the corresponding attributes

Distributed Systems

Deserialization

After deserialization the created object has the same state as the serialized original object (except for static and transient attributes)

transient attributes do get just standard values

As the return type of readObject is Object, we have to cast the reference to the generated object to the actual type (or one of its base classes)

Distributed Systems

Deserialization

If we write several objects to an ObjectOutputStream, we have to deserialize them in the same order

An attempt to deserialize an object can result in an error

For a call of readObject to be successful, it is necessary that:

- The next element of the input stream is indeed an object (and not a primitive data type)
- The object can be read completely and correct from the input stream

Deserialization

Additional prerequisites:

- A cast to the desired type has to be possible, i.e., the created object has to belong to the same class or a derived class
- The bytecode for the class to be deserialized has to be available. (The code for the class is not part of the serialized data)
- A class can be modified between serialization and deserialization. Therefore, an additional prerequisite is that the version of the class of the serialized object and the version of the class in the program that is deserializing the object have to be compatible
  - Java has a versioning mechanism for this purpose
  - such serialVersionUID can be generated by Java automatically or manually by the programmer

Example

```
import java.io.*;
public class ReadObjects {
    public static void main(String[] args) {
      try {
            FileInputStream fs = new FileInputStream("test.ser");
            ObjectInputStream is = new ObjectInputStream(fs);
            System.out.println("" + is.readInt());
            System.out.println((String)is.readObject());
            int i = is.readInt();
            System.out.println(i);
            String s  = (String )is.readObject();
            System.out.println(s);
            is.close();
      } catch (ClassNotFoundException e) { System.err.println(e.toString());
      } catch (IOException e) { System.err.println(e.toString()); }
    }
}
```


Object-References and Serialization

The serialization of objects is far from being trivial as objects can have attributes that refer to other objects

These referenced objects will be serialized, too

An algorithm for serialization therefore has to take care of the following tasks:

- following object-references
- correct handling of cyclic references
- reconstruction of the references from their serialized representation