Introduction to Java for C++ Programmers

Java Input / Output
Segment 2- Serialization

Professor: Mahboob Ali

Input Stream / Output Stream

In this segment you will be learning about:

- Serializable Objects
- FileInputStream / FileOutputStream
- ObjectInputStream / ObjectOutputStream

Object Serialization

- **Serialization** is a mechanism in java for writing a *state of the object into a byte stream.*
- **Deserialization** is the reverse operation of it.

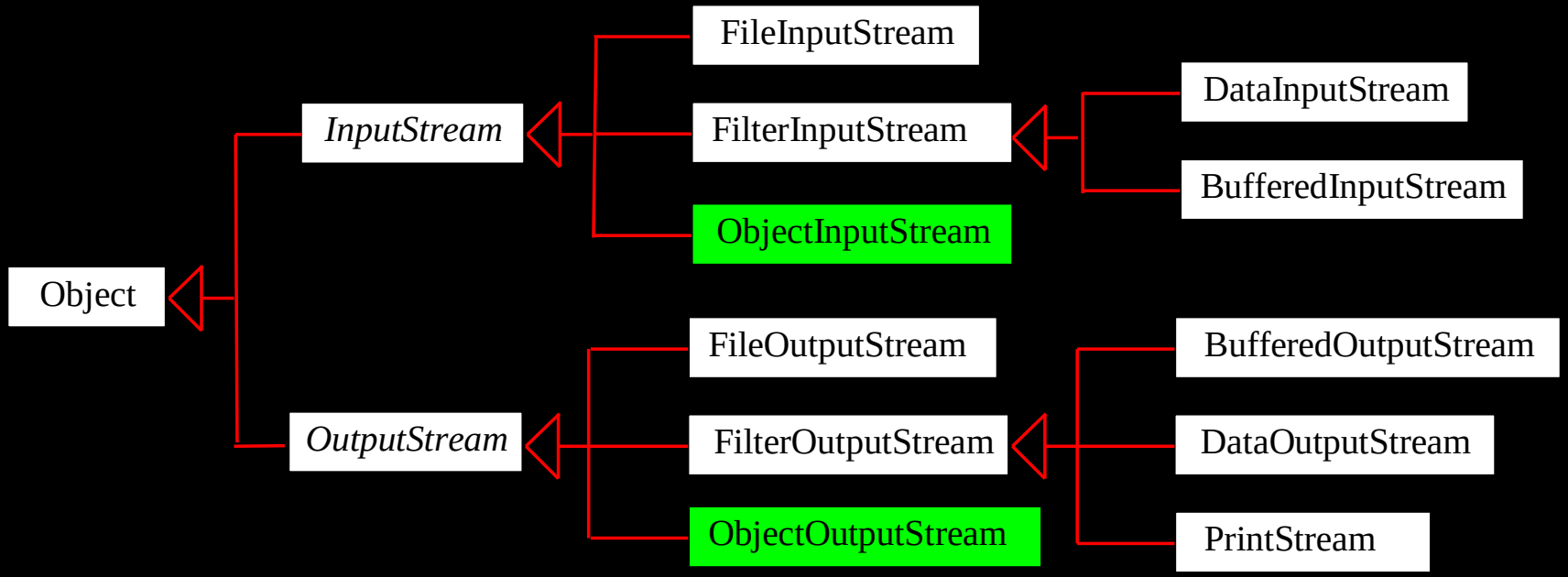
Advantage:

- Object serialization is necessary to save a state of an object into a disk file for *persistence.*
- Sent the object across the network for applications such as Web Services, Distributed-object applications, and Remote Method Invocation (RMI).

How to achieve Serialization?

- To make a Java object serializable we implement the **java.io.Serializable** interface.
- Serializable is a **marker interface** (has no data member and method). It is used to "mark" java classes so that objects of these classes may get certain capability.
- Other marker interfaces are **Cloneable** and **Remote.**

Object I/O



- DataInputStream/DataOutputStream enables you to perform I/O for primitive type values and strings. ObjectInputStream/ObjectOutputStream enables you to perform I/O for objects in addition for primitive type values and strings.

Using Object Streams

You may wrap an ObjectInputStream/ObjectOutputStream on any InputStream/OutputStream using the following constructors:

```
// Create an ObjectInputStream
public ObjectInputStream(InputStream in)

// Create an ObjectOutputStream
public ObjectOutputStream(OutputStream out)
```

```
import java.io.*;
public class ObjectInputOutputStream {
public static void main(String[] args) throws IOException,
                                    ClassNotFoundException {

try(ObjectOutputStream output = new ObjectOutputStream(new
                            FileOutputStream("object.dat"));)
       {
       output.writeUTF("John");
       output.writeDouble(85.5);
       output.writeObject(new java.util.Date());

       }
try (ObjectInputStream input = new ObjectInputStream(new
                            FileInputStream("object.dat"));)
       {
       String name = input.readUTF();
      double score = input.readDouble();
      java.util.Date date =
       (java.util.Date)(input.readObject());
      System.out.println(name + " " + score + " " + date);
       }
   }
}
```

The `Serializable` Interface

- Not all objects can be written to an output stream. Objects that can be written to an object stream is said to be *serializable*.
- A serializable object is an instance of the java.io.Serializable interface. So the class of a serializable object must implement Serializable.
- Implementing this interface enables the Java serialization mechanism to automate the process of storing the objects and arrays.

The `transient` Keyword

- If an object is an instance of Serializable, but it contains non-serializable instance data fields like static variables, can the object be serialized?
- The answer is no.
- To enable the object to be serialized, you can use the transient keyword to mark these data fields to tell the JVM to ignore these fields when writing the object to an object stream.

The `transient` Keyword, cont.

Consider the following class:

```
public class Foo implements java.io.Serializable {
  private int v1;
  private static double v2;
  private transient A v3 = new A();
}
class A { } // A is not serializable
```

When an object of the Foo class is serialized, only variable v1 is serialized. Variable v2 is not serialized because it is a static variable, and variable v3 is not serialized because it is marked transient. If v3 were not marked transient, a java.io.NotSerializableException would occur.

1. If a parent class has implemented Serializable interface then child class doesn't need to implement it but vice-versa is not true.
2. Only non-static data members are saved via Serialization process.
3. Static data members and transient data members are not saved via Serialization process. So, if you don't want to save value of a non-static data member then make it transient.
4. Constructor of object is never called when an object is de-serialized.
5. Associated objects must be implementing Serializable interface.

Random Access File

- All of the streams you have used so far are known as *read-only* or *write-only* streams.
- The external files of these streams are *sequential* files that cannot be updated without creating a new file.
- It is often necessary to modify files or to insert new records into files.
- Java provides the RandomAccessFile class to allow a file to be read from and write to at random locations.

File Pointer

- A random access file consists of a sequence of bytes.
- *File pointer:* a special marker that is positioned at one of these bytes.
- A read or write operation takes place at the location of the file pointer.
- When a file is opened, the file pointer sets at the beginning of the file.
- When you read or write data to the file, the file pointer moves forward to the next data.
- For example, if you read an int value using readInt(), the JVM reads four bytes from the file pointer and now the file pointer is four bytes ahead of the previous location.

Methods of the class

- Many methods in *`RandomAccessFile`* are the same as those in `DataInputStream` and `DataOutputStream`.
- For example:
  - `readInt()`,
  - `readLong()`,
  - `writeDouble()`,
  - `readLine()`,
  - `writeInt()`,
  - `writeLong()`.
- `void seek(long pos) throws IOException;`

  Sets the offset from the beginning of the `RandomAccessFile` stream to where the next read or write occurs.

Methods of the class

- `long getFilePointer() IOException;`

  Returns the current offset, in bytes, from the beginning of the file to where the next read or write occurs.

- `long length()IOException`

  Returns the length of the file.

- `final void writeChar(int v) throws IOException`

  Writes a character to the file as a two-byte Unicode, with the high byte written first.

- `final void writeChars(String s) throws IOException`

  Writes a string to the file as a sequence of characters.

Class's Constructors

- `RandomAccessFile raf = new RandomAccessFile("test.dat", "rw");`
- `// allows read and write`

- `RandomAccessFile raf = new RandomAccessFile("test.dat", "r");`
- `// read only`