City University

SE 409,410: Advanced Enterprise Java and Laboratory

Lecture 5

java IO and Serialization

Supta Richard Philip

supta.philip@gmail.com

---

java IO Package

- Java IO API is using to reading and writing data (input and output) from a file or over network, and write to a file or write a response back over the network.
- The Java IO API is located in the Java IO package (java.io).
- Java's IO package mostly concerns itself with the reading of raw data from a source and writing of raw data to a destination.
- The most typical sources and destinations of data are these:

> *Files*
>
> *Pipes*
>
> *Network Connections*
>
> *In-memory Buffers (e.g. arrays)*
>
> *System.in, System.out, System.error*

Streams

- IO Streams are a core concept in Java IO.
- A stream is a conceptually endless flow of data.
- Data can be read from a stream or write to a stream. A stream is connected to a data source or a data destination.
- Streams in Java IO can be either **byte based (reading and writing bytes) or character based (reading and writing characters)**
- A program that needs to read data from some source needs an **InputStream(reading bytes)) or a Reader(reading characters)**.
- A program that needs to write data to some destination needs an **OutputStream(writing bytes) or a Writer(writing characters)**.
- In Java, 3 streams are created for us automatically. All these streams are attached with the console.

```
1) System.out: standard output stream
2) System.in: standard input stream
3) System.err: standard error stream
```

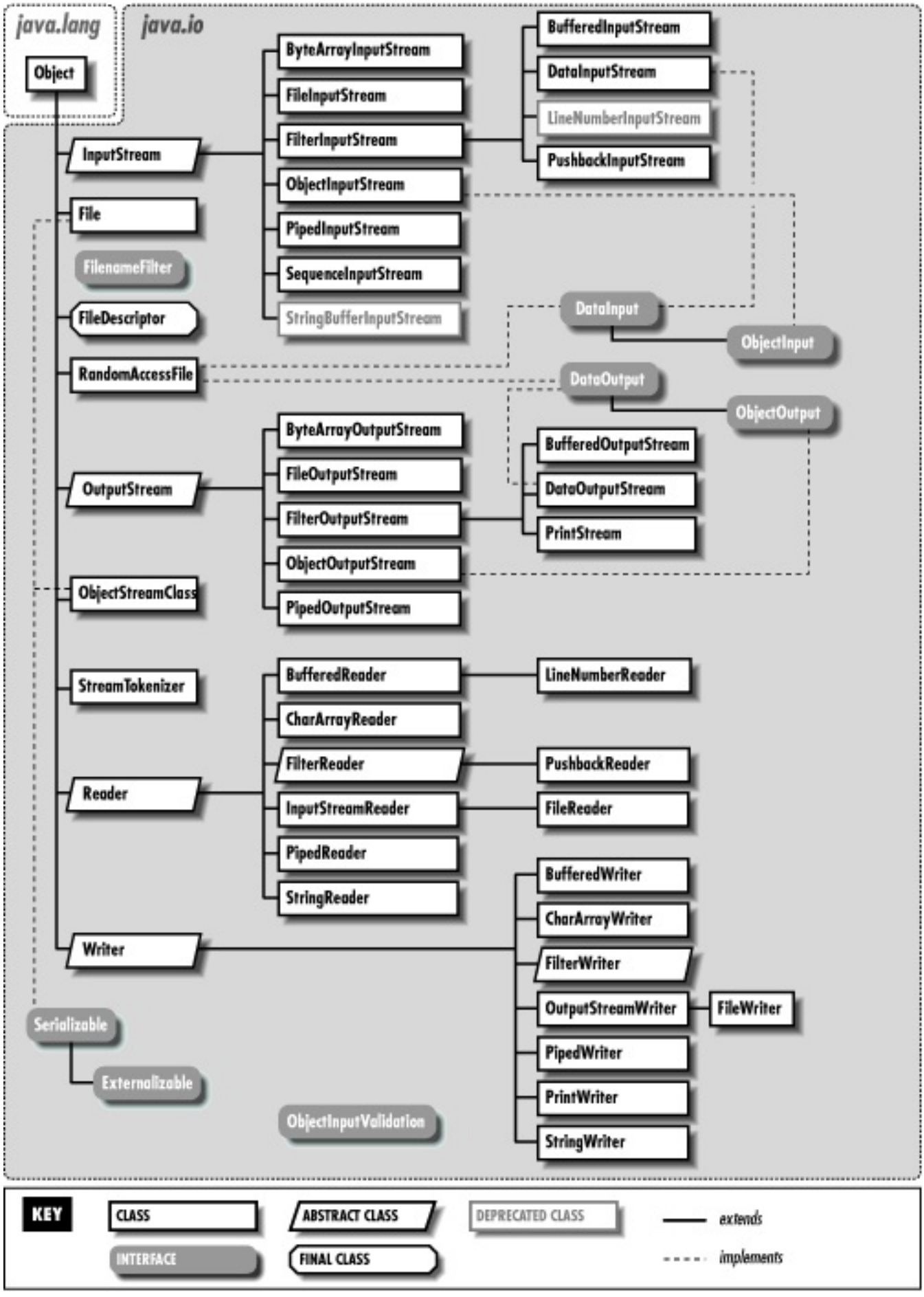
```
System.out.println("simple message");
System.err.println("error message");
```

Java IO Purposes

- Java IO contains many subclasses of the InputStream, OutputStream, Reader and Writer **abstract classes**.
- The **Reader and Writter** abstract classes read and write **16-bit unicode characters**.
- The **InputStream and OutputStream** abstract classes read and write **8-bit bytes**.
- Derived classes from the avobe 4 abstract classes add additional responsibilities using **decorator pattern**.
- All of these subclasses are addressing various different purposes using many different classes. The purposes addressed are summarized below:

> *File Access*
> *Network Access*
> *Internal Memory Buffer Access*
> *Inter-Thread Communication (Pipes)*
> *Buffering*
> *Filtering*
> *Parsing*
> *Reading and Writing Text (Readers / Writers)*
> *Reading and Writing Primitive Data (long, int etc.)*
> *Reading and Writing Objects.*

java IO Package structure


java.lang
java.io
Object
InputStream
File
FilenameFilter
FileDescriptor
RandomAccessFile
OutputStream
ObjectStreamClass
StreamTokenizer
Reader
Writer
Serializable
Externalizable
ByteArrayInputStream
FileInputStream
FilterInputStream
ObjectInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
BufferedInputStream
DataInputStream
LineNumberInputStream
PushbackInputStream
DataInput
ObjectInput
DataOutput
ObjectOutput
ByteArrayOutputStream
FileOutputStream
FilterOutputStream
ObjectOutputStream
PipedOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
BufferedReader
CharArrayReader
FilterReader
InputStreamReader
PipedReader
StringReader
LineNumberReader
PushbackReader
FileReader
BufferedWriter
CharArrayWriter
FilterWriter
OutputStreamWriter
PipedWriter
PrintWriter
StringWriter
FileWriter
ObjectInputValidation
KEY
CLASS
ABSTRACT CLASS
DEPRECATED CLASS
extends
INTERFACE
FINAL CLASS
implements

Java IO: File

- The File class in the Java IO API gives you access to the underlying file system.
- The File only gives you access to the file and file system meta data(directories as well).
- If you need to read or write the content of files, you should do so using either FileInputStream, FileOutputStream or RandomAccessFile.
- Before you can do anything with the file system or File class, you must obtain a File instance.

```
File file = new File("c:\\data\\input-file.txt");
```

- To check if the file exists, call the exists() method.

```
File file = new File("c:\\data\\input-file.txt");
boolean fileExists = file.exists();
.....................
File file = new File("javaFile.txt");
    if (file.createNewFile()) {
        System.out.println("New File is created!");
    } else {
        System.out.println("File already exists.");
    }
```

- The mkdir() method creates a single directory if it does not

already exist.

- The mkdirs() will create all directories that are missing in the path the File object represents.

```
    File file = new File("c:\\users\\jakobjenkov\\newdir")
;
    boolean dirCreated = file.mkdir();
    boolean dirCreated = file.mkdirs();
```

- To read the length of a file in bytes, call the length() method. object represents.

```
    File file = new File("c:\\data\\input-file.txt");
    long length = file.length();
```

- To rename (or move) a file, call the method renameTo() on the File class.

```
File file = new File("c:\\data\\input-file.txt");
boolean success = file.renameTo(new File("c:\\data\\new-fi
le.txt"));
```

- To delete a file call the delete() method.

```
   File file = new File("c:\\data\\input-file.txt");
   boolean success = file.delete();
```

- A File object can point to both a file or a directory.

```
    File file = new File("c:\\data");
    boolean isDirectory = file.isDirectory();
```

- The list() method returns an array of String's with the file and / or directory names of directory the File object points to. The listFiles() returns an array of File objects representing the files and / or directories in the directory the File points to.

```
    File file = new File("c:\\data");
    String[] fileNames = file.list();
    File[]   files = file.listFiles();
    //----------------------Example 1.....................
    File f=new File("C:\\Users\\CSE-12\\Desktop");
    String filenames[]=f.list();
    for(String filename:filenames){
        System.out.println(filename);
    }
    //.......... Example 2.....................
File dir=new File("C:\\Users\\CSE-12\\Desktop");
    File files[]=dir.listFiles();
    for(File file:files){
        System.out.println(file.getName()+" Can Write: "+fi
le.canWrite()+
        " Is Hidden:"+file.isHidden()+" Length: "+file.leng
th()+" bytes");
    }
```

File Input/Output Example 1

```
    File file = new File("hello.txt");
        //Print Writer
        PrintWriter out = new PrintWriter(file);
        out.println("Hello Richard");
        out.println(33);
        out.close();
        // read file data
        Scanner input = new Scanner(file);
        String name = input.nextLine();
        int age = input.nextInt();
        System.out.println("Name: "+name+" age: "+age);
        // Read keyboard input
        System.out.println("Enter your name:");
        Scanner sc = new Scanner(System.in);
        String name1 = sc.nextLine();
        System.out.println("Enter your age:");
        int age1 = sc.nextInt();
        System.out.println("name: "+name1+" age: "+age1);
```

Write object in file using Serialization

Student.java

```
import java.io.Serializable;
public class Student implements Serializable{
    private static final long serialVersionUID = 1L;
    private long id;
    private String name;
    private String address;
    private double salary;
    //getter,setetr,toString
```

SerializationDerializationFunction.java

```
package com.serialization;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
public class SerializationFunction {
    public static Object deSerialize(String file) throws I
OException, ClassNotFoundException {
        FileInputStream fileInputStream = new FileInputStr
eam(file);
        BufferedInputStream bufferedInputStream = new Buff
eredInputStream(fileInputStream);
        ObjectInputStream objectInputStream = new ObjectIn
```

```
putStream(bufferedInputStream);
        Object object = objectInputStream.readObject();
        objectInputStream.close();
        return object;
    }
public static void serialize(String file, Object object) t
hrows IOException {
        FileOutputStream fileOutputStream = new FileOutput
Stream(file);
        BufferedOutputStream bufferedOutputStream = new Bu
fferedOutputStream(fileOutputStream);
        ObjectOutputStream objectOutputStream = new Object
OutputStream(bufferedOutputStream);
        objectOutputStream.writeObject(object);
        objectOutputStream.close();
    }
}
```

App.java

```
import java.io.IOException;
public class App {
    public static void main(String args[]) {
        Student s = new Student();
        s.setId(1090);
        s.setName("Richard");
        s.setAddress("Dhaka");
```

```
        s.setSalary(50000);
        System.out.println(s);
        try {
            SerializationFunction.serialize("student.txt",
 s);
            Student student = (Student) SerializationFunct
ion.deSerialize("student.txt");
            System.out.println(student);
        } catch (IOException exp) {
            exp.printStackTrace();
        } catch (ClassNotFoundException exp) {
            exp.printStackTrace();
        }
    }
}
```

JAXB API for XML

> *Employee POJO class with XML annotation.*

Employee.java

```
import java.io.Serializable;
import javax.xml.bind.annotation.*;
@XmlRootElement(name = "employee")
public class Employee {
    private String id;
```

```
    private String name;
    private String address;
    private int salary;
    public Employee() {
    }
    public Employee(String id, String name, String address
, int salary) {
        this.id = id;
        this.name = name;
        this.address = address;
        this.salary = salary;
    }
    @XmlElement
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    @XmlElement
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @XmlElement
    public String getAddress() {
```

```
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    @XmlElement
    public int getSalary() {
        return salary;
    }
    public void setSalary(int salary) {
        this.salary = salary;
    }
}
```

EmployeeJAXB.java

```
import java.io.File;
import javax.xml.bind.*;
public class EmployeeJAXB {
    public void marshall() {
        try {
            Employee emp = new Employee("A001", "rr", "Dha
ka", 2000);
            JAXBContext jc = JAXBContext.newInstance(Emplo
yee.class);
            Marshaller ms = jc.createMarshaller();
            ms.setProperty(Marshaller.JAXB_FORMATTED_OUTPU
```

```
T, true);
            ms.marshal(emp, System.out);
            ms.marshal(emp, new File("src//Employee.xml"))
;
        } catch (Exception e) {
            System.out.println(" " + e.getMessage());
        }
    }
    public void unmarshall() {
        try {
            JAXBContext jc = JAXBContext.newInstance(Emplo
yee.class);
            Unmarshaller ums = jc.createUnmarshaller();
            Employee emp = (Employee) ums.unmarshal(new Fi
le("src//Employee.xml"));
            System.out.println("Emp id " + emp.getId());
            System.out.println("Emp name " + emp.getName()
);
            System.out.println("Emp address " + emp.getAdd
ress());
            System.out.println("Emp salary " + emp.getSala
ry());
        } catch (Exception e) {
            System.out.println(" " + e.getMessage());
        }
    }
}
```

App.java

```
public class App {
    public static void main(String[] args) {
        EmployeeJAXB jaxb = new EmployeeJAXB();
        System.out.println("marshall started.....");
        jaxb.marshall();
        System.out.println("marshall done.....");
        System.out.println("..............");
        System.out.println("unmarshall started.....");
        jaxb.unmarshall();
        System.out.println("unmarshall done.....");
    }
}
```

References

https://www.youtube.com/watch?v=YirpEhdj0Nw&t=679s

http://tutorials.jenkov.com/java-io/outputstreamwriter.html