IO

Objectives

- Differentiate between 2 types of stream,
- Write programs to read, write and modify text files,
- Seralize java objects,

Table of Content

Working with files at OS level

- **`java.io.File`** class can be used to work with system dependent commands for files and directories.
- The path name in the code hence will depend on the underlying OS in which JVM is installed.
- To make the code portable so that it works on all systems, **`static`** member **`separator`** defined in the **`File`** class can be used.
- The path name can be either *absolute* or *relative*.
- The access permissions on a File object may cause some methods in this class to fail.
- Instances of the File class are immutable; that is, once created, the abstract pathname represented by a File object will never change!

Go through the File methods in API for 10 minutes

Example: Creating a file

```
import java.io.*;
class FileOper{
public static void main(String str[]){
try{
File file = new File("newFile.txt");
if(file.exists())
file.delete();
boolean b=file.createNewFile();
System.out.println(b);
}catch(IOException e){ }
}}
```

Exercise

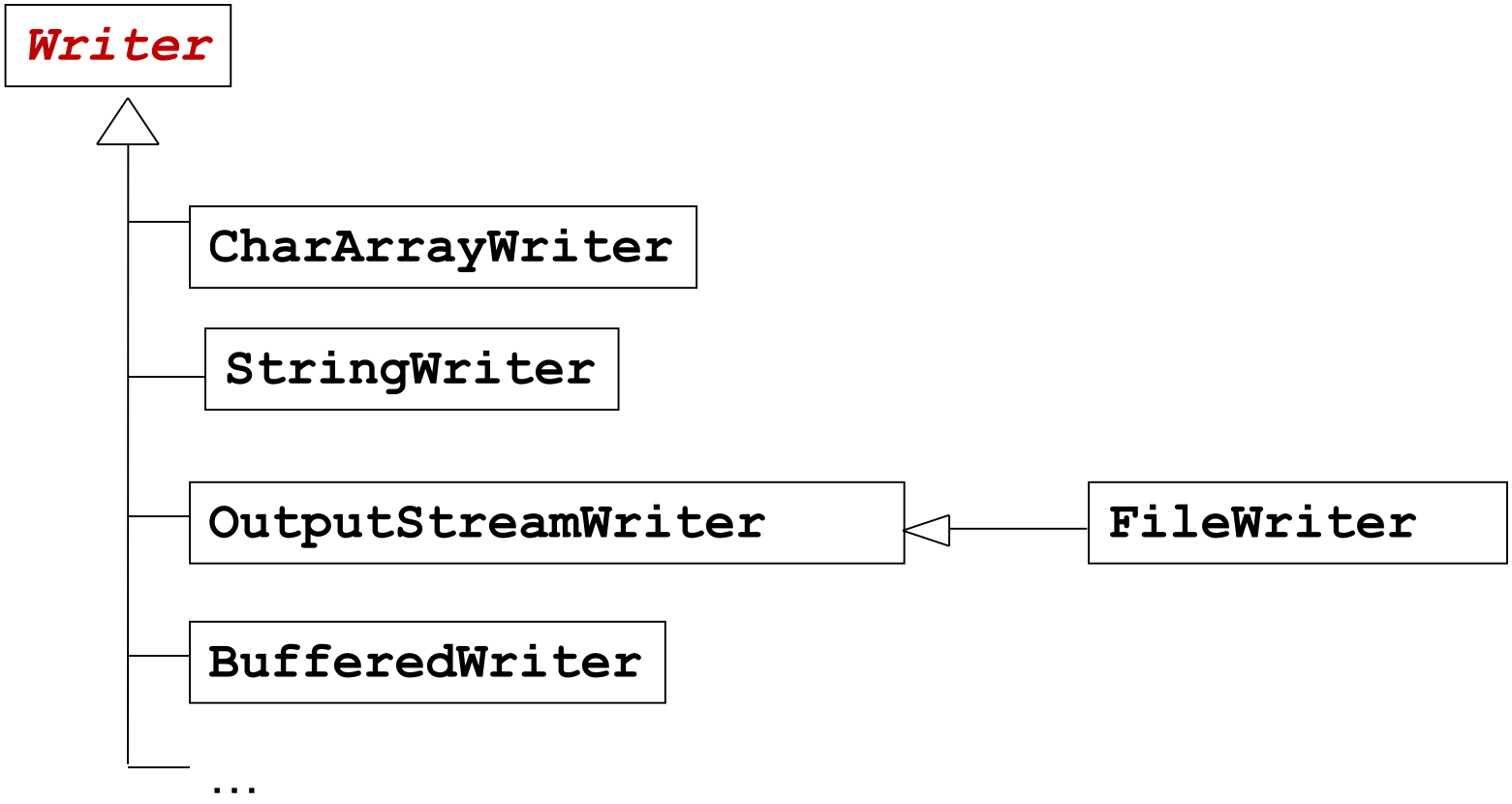
- *Write a program that creates a new file called BatchMates.txt and store it in a directory named Batch. Also list the files or subdirectories present in the drive/directory where the directory Batch exists, stating if it is a file or directory.*

(30 mins)

What are streams?

- An IO stream is an abstract term for any type of input or output device.
- Stream is a sequence of data
- Character stream
  - Character stream writer classes
  - Character stream reader classes
- Byte stream
  - Byte stream writer classes
  - Byte stream reader classes
  - Supports Serialization

Character stream



Writer

- It is an abstract class for writing to character streams.
- Methods are to write or append a character or character array or strings and flush.
- All the methods throw **`IOException`**.

Go through the methods of Writer class for 5 minutes.

FileWriter

- This class is used to create and write characters to the file.
- In some platforms, a file can be opened for writing by only one **`FileWriter`** at a time.

Go through the FileWriter API.

new FileWriter("x.txt");
What will happen if the file x.txt exists when the above statement executes?

How can you write at the end of the file?

Writing into a file using `FileWriter`

```
import java.io.*;
public class A{
public static void main(String args[])  {
FileWriter f= null;
try{
      f =new
FileWriter("D:"+File.separator+"register.txt");
      char c[]= args[0].toCharArray();
      f.write("Hello ");
      f.write(c,0,c.length);
      f.write("\n");

} catch(IOException ioe){}
finally{
      try{  if(f!=null)f.close();}
      catch(IOException e){}}
 }}
```

```
F:\JavaTest>java A Rana
```

BufferedWriter

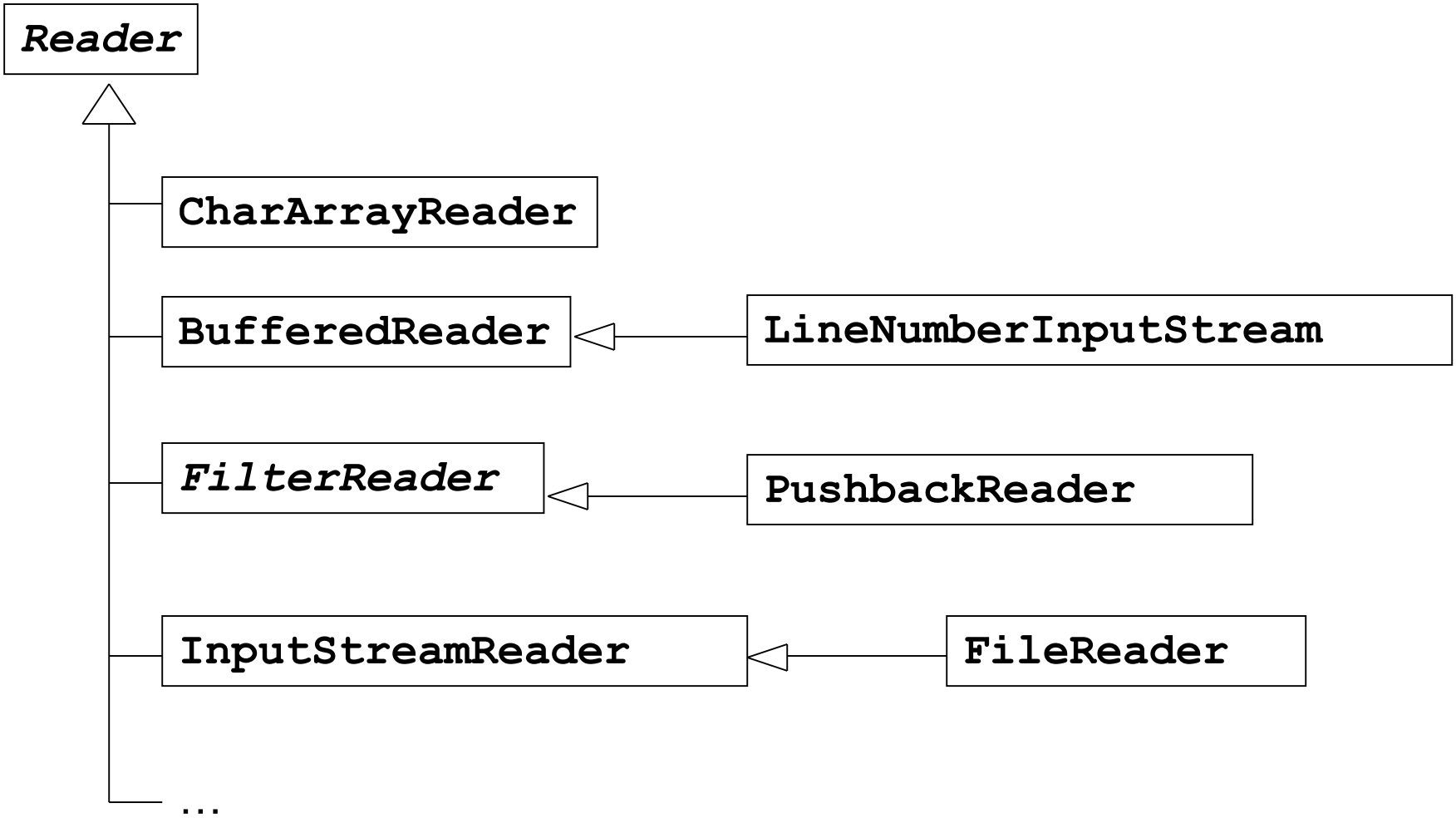
- This class wraps the Writer class to provide additional functionality of buffering characters for the efficient writing of single characters, arrays, and strings.
- It adds a method **`void newLine() throws IOException`** that is very handy to write lines into the text files.

Exercise

Accept the roll no., name and grade for 3 students and write each student details in a separate line in a file named student.csv. The details must be separated by comma.

(30 mins)

Hierarchy of character stream reader



Reader

- **`Reader`** is an abstract class for reading character streams.
- It has methods to read characters and also methods like mark and reset which are used to position file pointers appropriately.
- It is not compulsory for all the classes inheriting from the **`Reader`** to support mark and reset methods.
- The inheriting classes that do not support this method return false when **`markSupported()`** is called.

Go through the Reader class in the API

How does the mark and reset methods work?

FileReader

- **`FileReader`** is class used to read characters from a file.
- This class does not add any new methods.
- If the file specified in the constructor for opening is not available, **`FileNotFoundException`** is thrown.
- **`FileNotFoundException`** is a subclass of **`IOException`**

Go through the FileReader class in the API

BufferedReader

- Reads text from a character-input stream by buffering characters for the efficient reading of characters, arrays, and lines.

Constructor:

```
BufferedReader(Reader in)
```

```
BufferedReader(Reader in, int sz)
```

The default buffer size is large enough for most purposes. In certain cases where more size is required, a size value can be specified.

Methods:

```
String readLine() throws IOException
```

This class supports **`mark()`** and **`reset()`**

PushbackReader

- This class allows characters to be pushed back into the stream. This is a wrapper class.
- This class supports **`mark()`** and **`reset()`**
- Constructor
  - **`PushbackReader(Reader in)`**
- Methods
  - **`void unread(int c)`**

    Pushes back a character specified by c by copying it to the front of the pushback buffer. Next character that will be read is c.
  - **`void unread(char[] cbuf)`**
  - **`void unread(char[] cbuf, int off, int len)`**

    Pushes back a char array or part of char array (of length **`len`** starting from **`offset`** off) by copying it to the front of the pushback
  - **`long skip(long n)`**

    Places the file pointer after n characters.

Exercise

- *Ram wrote a Java code to create a text file that will store many file paths accessible by a large application with its timestamp. This code went live and later it was found that the file had the required data but instead of a new line separator between each network path, it had \n.*

  Path.txt

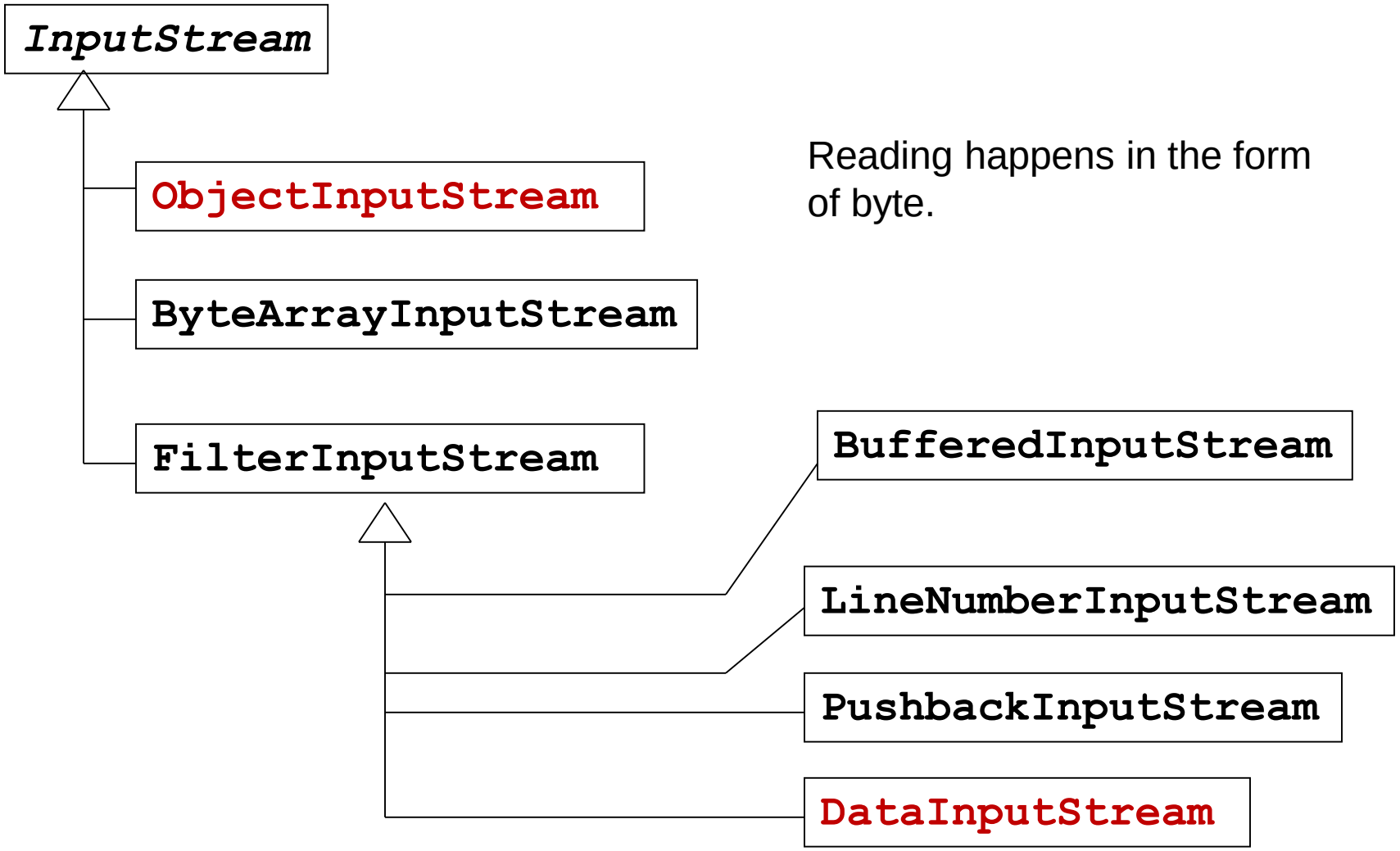
  2012-1-30 T 10:45 UTC - E:\tomat \n 2012-1-30 T 12:45 UTC F:\ Data \n 2012-2-30 T 2:45 UTC - E:\MySQL

- *Write java code that reads and replaces \n by new line character and write it to the file. Note that since the data in the file is very large , reading string data at one stroke may lead to overflow problems.*

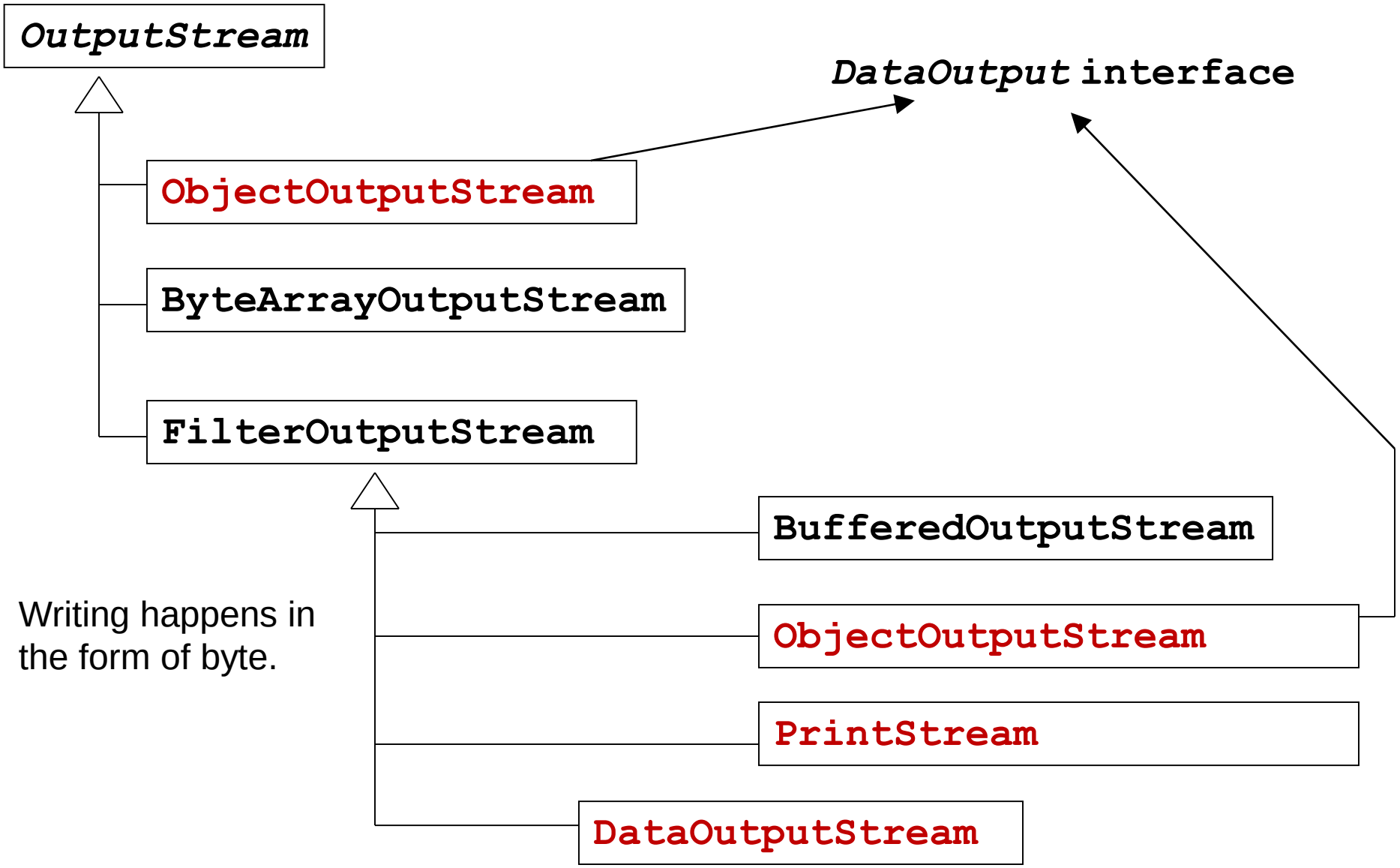
  Hint : you may need to read and unread characters

(45 mins)

Hierarchy of byte stream



Reading happens in the form of byte.



Writing happens in the form of byte.

Exercise

Go through ***`FileInputStream and FileOutputStream API`***

- *Write a program using byte stream that copies the content of one file into another file.*

(45 mins)

PrintStream

- **`PrintStream`** is a class that has a functionality with the ability to print representations of various data values conveniently.
- **`System.out`** is an instance of **`PrintStream`**.
- Apart from this, two other functionalities that are provided here are:

A) Unlike other output streams, a **`PrintStream`** never throws an **`IOException`**

B)The **`flush()`** method can be automatically invoked after **`println`** method is invoked or newline **`('\n')`** is written.

Go through the PrintStream API.

Which method in the **`PrintStream`** helps formatting the output?

Exercise

- *Create a file called product list. Prompt the user to enter the name, price and manufacturing date. Save formatted data in the product list file so that name of the product should not exceed 10 characters, price is a double saved with precision as 2 chars after decimal such as 100.00 and date in format 05/29/06.*

(45 mins)

Serialization

- The mechanism of storing the state of an object in the hard disk so that it can be restored later by your program.
- Serialization enables storing values of all instance variables which includes both primitives and **`Serializable`** objects.
- Serialization mechanism creates a file into which the state of the object is written.
- This file can later be read by the java program which can then restore the object's state.
- **`ObjectOutputStream`** and **`ObjectInputStream`** classes are used for these purposes. They are wrapper classes that take **`OutputStream`** and ***`InputStream`*** objects respectively

Serialization classes

- **`ObjectOutputStream`**
  - **`ObjectOutputStream(OutputStream out)throws IOException`**
  - **`void writeXxx(xxx v)`** where xxx is any primitive type, or **`Object`**
  - **`void write(int x) out)throws IOException`**
  - And all the methods from **`OutputStream`**
- **`ObjectInputStream`**
  - **`ObjectInputStream(InputStream in) throws IOException`**
  - **`xxx readXxx() out)throws IOException`** where xxx is any primitive type, or **`Object`**

    `readObject()throws ClassNotFoundException` also in addition to **`IOException`**
  - **`int read()`**
  - And all the methods from **`InputStream`**

Steps to save and retrieve an object's state

Saving an object state

```
1.  FileOutputStream f= new
    FileOutputStream("MySerFile.ser");
2.  ObjectOutputStream obfile= new
    ObjectOutputStream(f);
3.   obfile.writeObject(objectInstance);
4.  Obfile.close();
```

Retrieving an object state

```
1.  FileInputStream f= new
    FileInputStream("MySerFile.ser");
2.  ObjectInputStream obfile= new
    ObjectInputStream(f);
3.  Object o=obfile.readObject();
4.  MyObject m=(MyObject)o;
5.  Obfile.close();
```

java.io.Serializable

- Only the objects which implement **Serializable** interface can be serialized.

  class MyObject implements Serializable{… }

- **Serializable** is a marker interface.
- If object has references, then the references also must be either **Serializable** or should be marked **transient**.
- In JSE, some classes are not **Serializable.** For example **Thread class, Subclasses of Writer, Reader, InputStream, OutputStream.**
- All the collection classes, all primitive wrappers**, String, StringBuffer, StringBuilder** are **Serializable**
- If an attempt to serialize an object that does not implement **Serializable** is made, **NotSerializableException** is thrown.

transient

- Instance variables marked **`transient`** will not be saved.
- When object is de-serialized the **`transient`** variables are set to the default value based on their type.
- During serialization even the **`private`** state of the object is stored.
- Hence sensitive information like credit card number, password, a file descriptor contains a handle that provides access to an operating system resource must be marked transient.
- Also if a class contains references of object that cannot be serialized (like Thread), must be marked **`Serializable`** .

Example: Serialization

```
package general;

public abstract class Person
implements Serializable{

…

}

import java.io.*;

public class SerializeP {

public static void main(String str[]) throws
   IOException{

 Teacher f=new Teacher ("Tom");
```

```
//saving Teacher
ObjectOutputStream o=
            new   ObjectOutputStream(
            new   FileOutputStream("t.ser"));
 o.writeObject(f);
  o.close();
// reloading the object state from file
ObjectInputStream in= new ObjectInputStream(
                  new FileInputStream("t.ser"));
 f=(Teacher )in.readObject();
 System.out.println(f);
in.close();
}}
```

Could be any extension

Beware!

- You could save any number of objects in a file. The definition of **`readObject`** doesn't specify that it will return null when the end of stream is reached. Instead an exception is thrown if you attempt to read an additional object beyond the end of the file.
- Care must be taken while de-serializing the objects.

1. The objects must be cast into its correct type otherwise an **`ClassCastException`** will be thrown at runtime
2. The objects must be retrieved in the same way as they are saved. For instance, if you save an integer using **`writeInt()`** then you must retrieve using **`readInt()`** method. Using **`readObject()`** and casting it back to **`int`** will not work(an **`java.io.OptionalDataException`** will be thrown at runtime)

- Safest and more common way to save and retrieve is to use **`writeObject()`** and **`readObject()`** methods.
- **`readObject()`** and **`writeObject()`** are only for non-static and non-transient fields

Exercise

- *Create an object called employee whose attributes are emp_id, emp_name and emp_sal. Write a program to Serialize and deserialize the employee object except for the emp_sal attribute.*

(30 mins)

Best practices

- Always close streams
- Opt for buffering while reading or writing to streams

Summary

- java.io.File class can be used to work with system dependent commands for files and directories. Instances of the File class are immutable.
- Writer is an abstract class for writing to character streams and Reader is an abstract class for reading character streams.
- FileWriter class is used to create and write characters to the file and FileReader is class used to read characters from a file.
- BufferedWriter class wraps the Writer for buffering characters and BufferedReader reads text from a input stream by buffering characters.
- PushbackReader class allows characters to be pushed back into the stream.
- PrintStream class prints the representations of various data values.
- Serialization is the mechanism of storing the state of an object in the hard disk. The objects which implement Serializable interface can be serialized.