Streams and Input/Output Files
Part 2

Files and Exceptions

- When creating files and performing I/O operations on them, the systems generates errors. The basic I/O related exception classes are given below:
  - EOFException – signals that end of the file is reached unexpectedly during input.
  - FileNotFoundException – file could not be opened
  - InterruptedIOException – I/O operations have been interrupted
  - IOException – signals that I/O exception of some sort has occurred – very general I/O exception.

Syntax

- Each I/O statement or a group of I/O statements must have an exception handler around it/them as follows:

```
try {
…// I/O statements – open file, read, etc.
}
catch(IOException e) // or specific type exception
{
    …//message output statements
}
```

Example

```
import java.io.*;
class CountBytesNew {

    public static void main (String[] args)
        throws FileNotFoundException, IOException  // throws is optional in this case
    {

        FileInputStream in;
        try{
            in = new FileInputStream("FileIn.txt");
            int total = 0;
            while (in.read() != -1)
                total++;
            System.out.println("Total = " + total);
        }
        catch(FileNotFoundException e1)
        {
            System.out.println("FileIn.txt does not exist!");
        }
        catch(IOException e2)
        {
            System.out.println("Error occured while read file FileIn.txt");
        }

    }

}
```

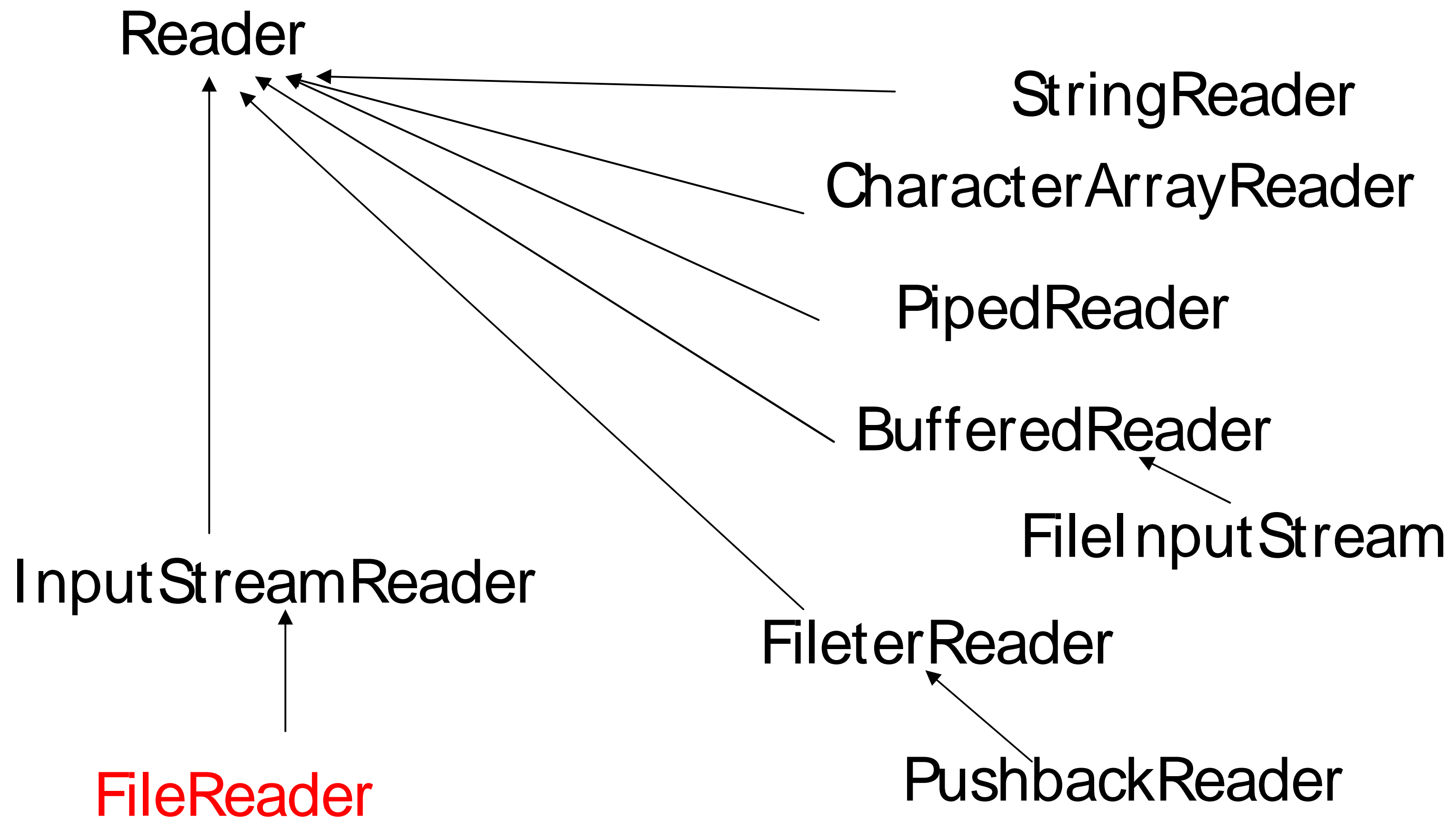
Creation of Files

- There are 2 ways of initialising file stream objects:
  - Passing file name directly to the stream constructor
    - Similar to previous example
  - Passing File Object:
    - Create File Object
      - File inFile = new File(**"FileIn.txt");**
    - **Pass file object while creating stream:**
      - **try {**
        - **in = new FileInputStream(**inFile**);**
      - **}**
- **Manipulation operations are same once the file is opened.**

Reading and Writing Characters

- As pointed out earlier, subclasses of Reader and Writer implement streams that can handle characters.
- The two subclasses used for handling characters in file are:
  - FileReader
  - FileWriter
- While opening a file, we can pass either file name or File object during the creation of objects of the above classes.

Reader Class Hierarchy



Reader - operations

public int read()	Reads a character and returns as a integer 0-255
public int read(char[] buf, int offset, int count)	Reads and stores the characters in *buf* starting at *offset*. *count* is the maximum read.
public int read(char[] buf)	Same as previous *offset*=0 and *length*=*buf.length*()
public long skip(long count)	Skips *count* characters.
public boolean()	Returns true if the stream is ready to be read.
public void close()	Closes stream

Reader - example

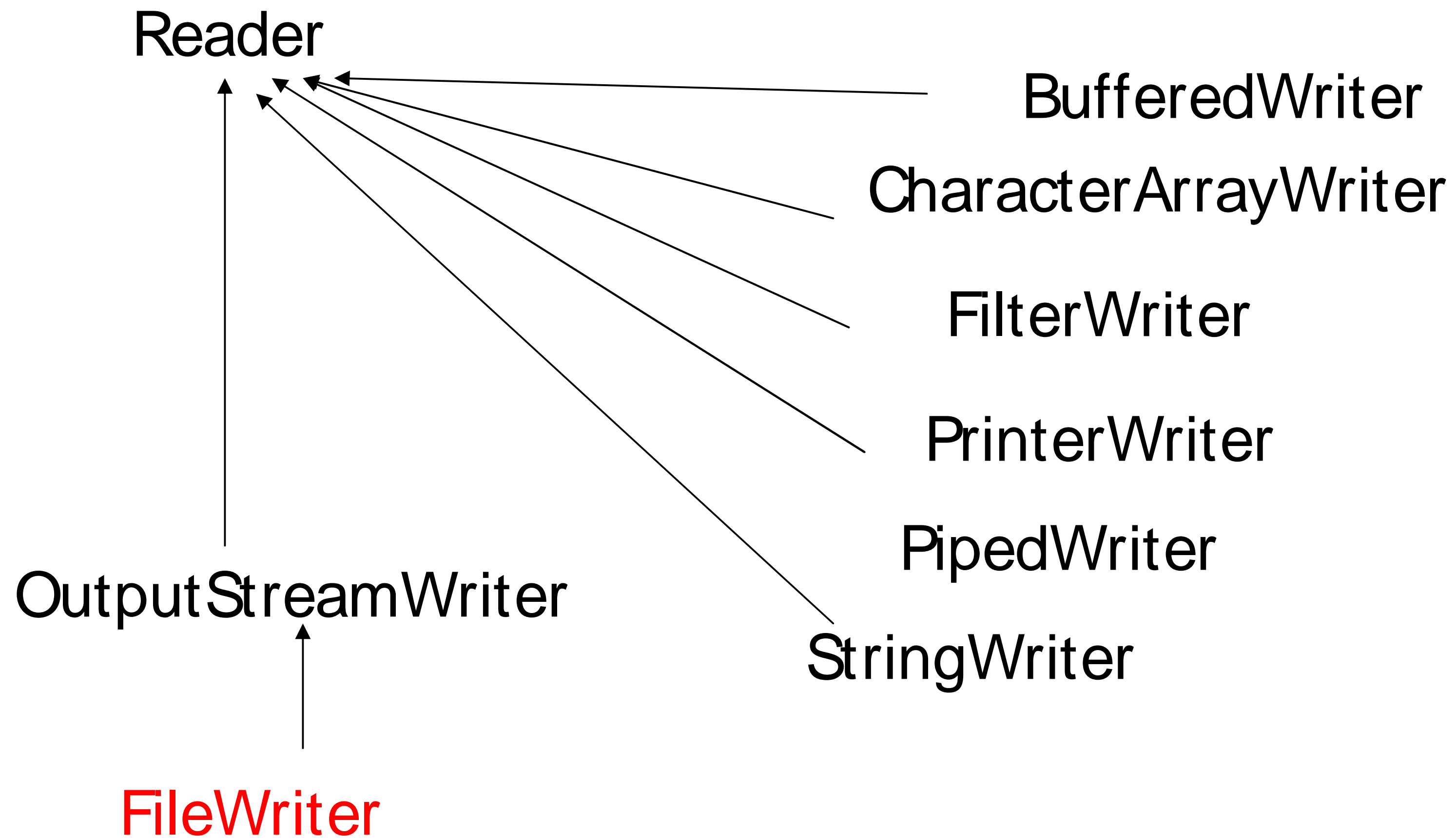
- Count total number of spaces in the file

```
import java.io.*;
public class CountSpace {
	public static void main (String[] args)
		throws IOException
	{
		Reader in;  // in can also be FileReader
		in = new FileReader("FileIn.txt");
		int ch, total, spaces;

		spaces = 0;

		for (total = 0 ; (ch = in.read()) != -1; total++){
			if(Character.isWhitespace((char) ch))
			{
				spaces++;
			}
		}
		System.out.println(total + " chars " + spaces + " spaces ");
	}
}
```

Writer Class Hierarchy



Byte Output Streams - operations

public abstract void write(int ch)	Write *ch* as characters.
public void write(char[] buf, int offset, int count)	Write *count* characters starting from *offset* in *buf.*
public void write(char[] buf)	Same as previous *offset=0* and *count = buf.length()*
public void write(String str, int offset, int count)	Write *count* characters starting at *offset* of *str.*
public void flush()	Flushes the stream.
public void close()	Closes stream

Copying Characters from Files

- Write a Program that copies contents of a source file to a destination file.
- The names of source and destination files is passed as command line arguments.
- Make sure that sufficient number of arguments are passed.
- Print appropriate error messages.

FileCopy.java

```
import java.io.*;
public class FileCopy {

    public static void main (String[] args)
    {
        if(args.length != 2)
        {
            System.out.println("Error: in sufficient arguments");
            System.out.println("Usage - java FileCopy SourceFile DestFile");
            System.exit(-1);
        }
        try {
        FileReader srcFile = new FileReader(args[0]);
        FileWriter destFile = new FileWriter(args[1]);

        int ch;
        while((ch=srcFile.read()) != -1)
            destFile.write(ch);
        srcFile.close();
        destFile.close();
        }
        catch(IOException e)
        {
            System.out.println(e);
            System.exit(-1);
        }
    }
}
```

Runs and Outputs

- Source file exists:
  - java FileCopy FileIn.txt Fileout.txt
- Source file does not exist:
  - java FileCopy abc Fileout.txt

    java.io.FileNotFoundException: abc (No such file or directory)
- In sufficient arguments passed
  - java FileCopy FileIn.txt

    Error: in sufficient arguments

    Usage - java FileCopy SourceFile DestFile

Buffered Streams

- Buffered stream classes – BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter buffer data to avoid every read or write going to the stream.

- These are used in file operations since accessing the disk for every character read is not efficient.

Buffered Streams

- Buffered character streams understand lines of text.
- BufferedWriter has a newLine method which writes a new line character to the stream.
- BufferedReader has a readLine method to read a line of text as a String.
- For complete listing of methods, please see Java documentation.

BufferedReader - example

- Use a BufferedReader to read a file one line at a time and print the lines to standard output

```
import java.io.*;

class ReadTextFile {
        public static void main(String[] args)
                throws FileNotFoundException, IOException
        {
                BufferedReader in;
                in = new BufferedReader( new FileReader(“Command.txt”));
                String line;
                while (( line = in.readLine()) != null )
                {
                        System.out.println(line);
                }
        }
}
```

Reading/Writing Bytes

- The FileReader and FileWriter classes are used to read and write 16-bit characters.
- As most file systems use only 8-bit bytes, Java supports number of classes that can handle bytes. The two most commonly used classes for handling bytes are:
  - FileInputStream (discussed earlier)
  - FileOutputStream

Writing Bytes - Example

```
public class WriteBytes {

    public static void main (String[] args)
    {
        byte cities[] = {'M', 'e', 'l', 'b', 'o', 'u', 'r', 'n', 'e', '\n', 'S', 'y','d', 'n', 'e', 'y', '\n' };

         FileOutputStream outFile;
         try{
               outFile = new FileOutputStream("City.txt");
               outFile.write(cities);
               outFile.close();
         }
         catch(IOException e)
         {
               System.out.println(e);
               System.exit(-1);
         }

    }

}
```

Summary

- All Java I/O classes are designed to operate with Exceptions.
- User Exceptions and your own handler with files to manger runtime errors.
- Subclasses FileReader / FileWriter support characters-based File I/O.
- FileInputStream and FileOutputStream classes support bytes-based File I/O.
- Buffered read operations support efficient I/O.