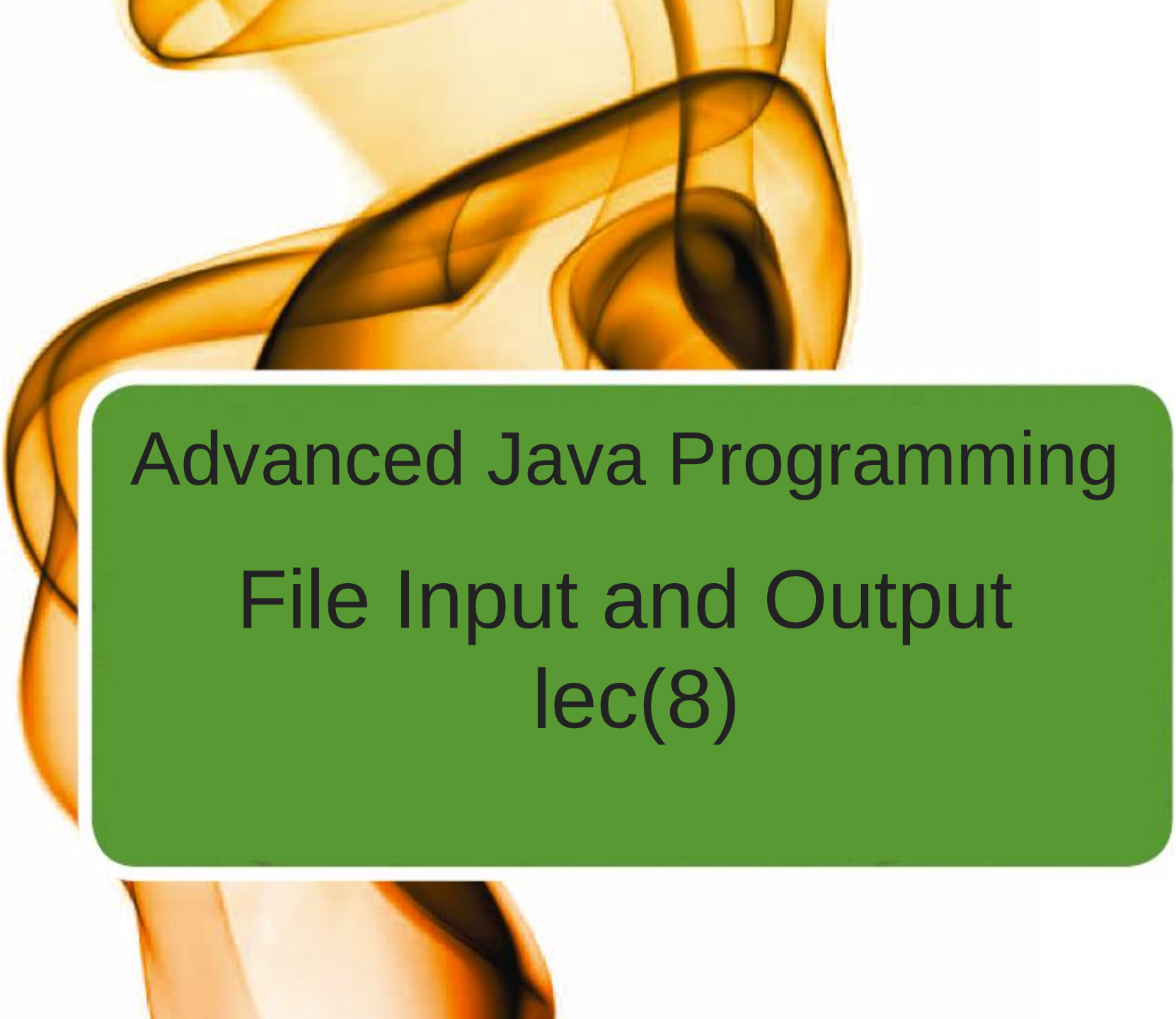


Advanced Java Programming

File Input and Output
lec(8)

File Organization, Streams, and Buffers (cont'd.)

- a Java application perform an input/output operation through a stream.
- **Stream is** a pipeline or channel.
- bytes flow in/out of your application through stream from/to an input/output device.

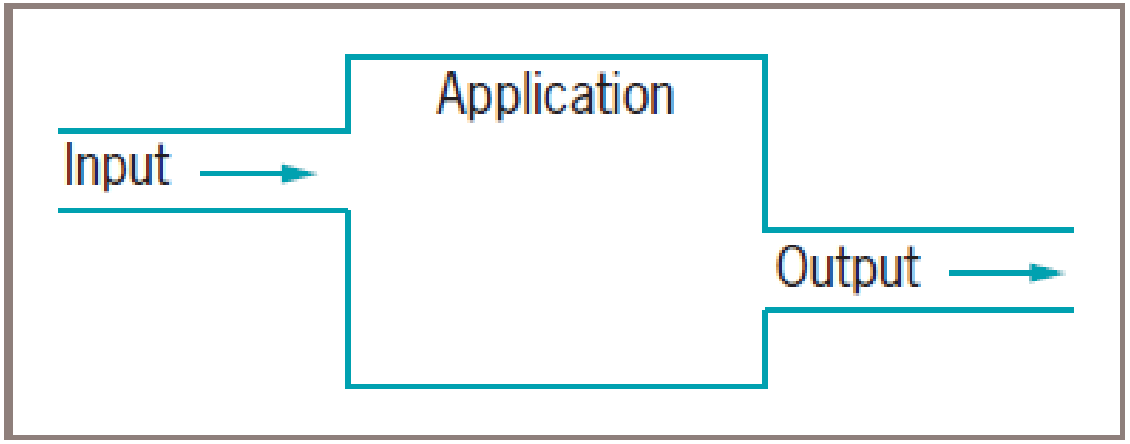


Figure 13-13 File streams

File Organization, Streams, and Buffers (cont'd.)

- Most streams flow in only **one direction**
- stream is either an input or output stream.
- Random access files use streams that flow in two directions.
- A stream is an **object**, having data and methods.
- The stream methods allow you to perform actions such as opening, closing, reading, and writing.

File Organization, Streams, and Buffers (cont'd.)

- **Buffer**
  - Is a Memory location.
  - Using a buffer to accumulate input or output before issuing the actual IO command improves program performance.
  - Why? because storage devices are usually slow.
- **Flushing**
  - Clears any bytes that have been sent to a buffer for output , but have not yet been output to a hardware device

Using Java's IO Classes

- Figure 13-14 shows a partial hierarchical relationship of some of the classes Java uses for input and output (IO) operations.
- `InputStream`, `OutputStream`, and `Reader`.
- All these classes are abstract, that contain methods for performing input and output.

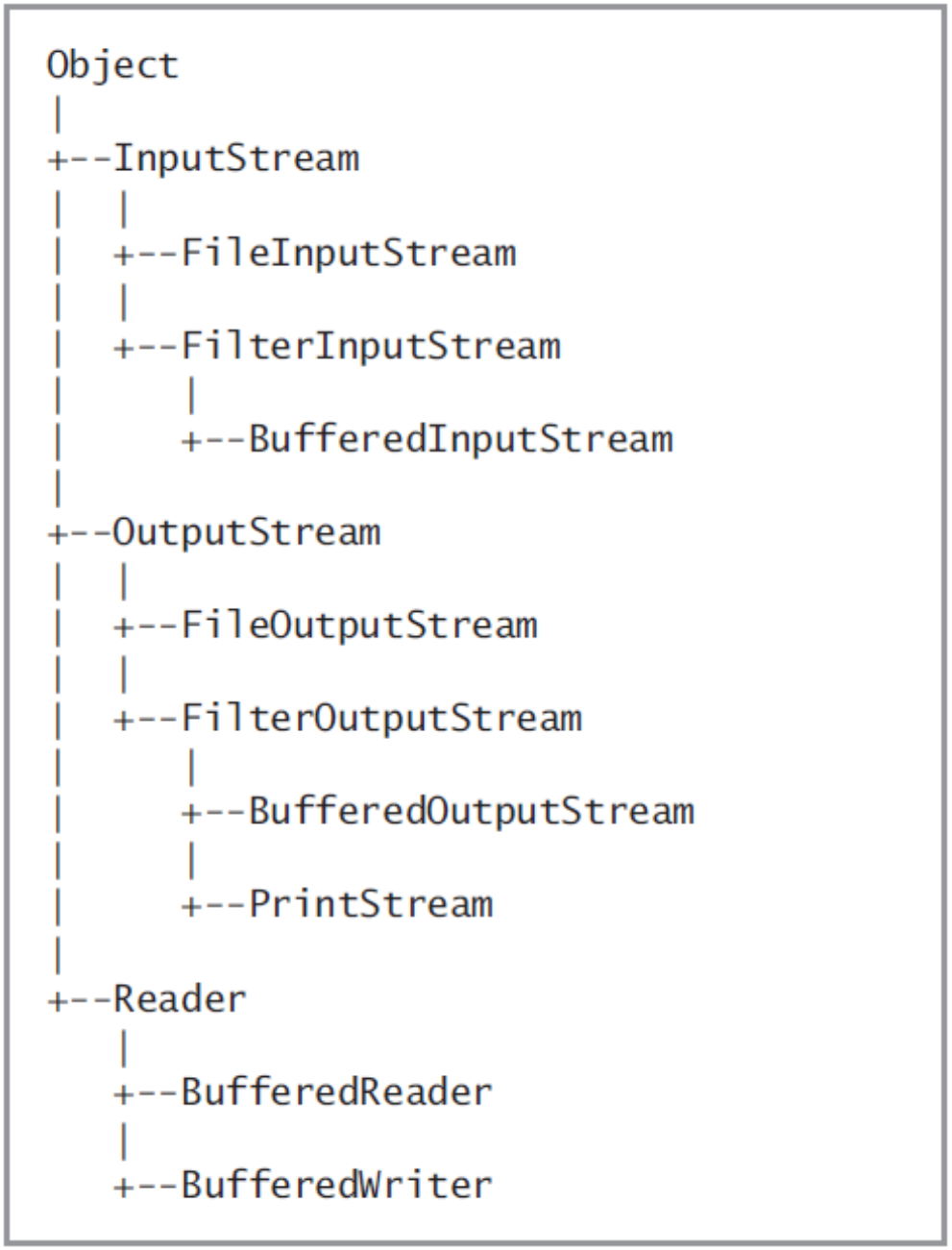


Figure 13-14 Relationship of selected IO classes

Using Java's IO Classes (cont'd.)

Class	Description
`InputStream`	Abstract class that contains methods for performing input
`FileInputStream`	Child of `InputStream` that provides the capability to read from disk files
`BufferedInputStream`	Child of `FilterInputStream`, which is a child of `InputStream`; `BufferedInputStream` handles input from a system's standard (or default) input device, usually the keyboard
`OutputStream`	Abstract class that contains methods for performing output
`FileOutputStream`	Child of `OutputStream` that allows you to write to disk files
`BufferedOutputStream`	Child of `FilterOutputStream`, which is a child of `OutputStream`; `BufferedOutputStream` handles input from a system's standard (or default) output device, usually the monitor
`PrintStream`	Child of `FilterOutputStream`, which is a child of `OutputStream`; `System.out` is a `PrintStream` object
`Reader`	Abstract class for reading character streams; the only methods that a subclass must implement are `read(char[], int, int)` and `close()`
`BufferedReader`	Reads text from a character-input stream, buffering characters to provide for efficient reading of characters, arrays, and lines
`BufferedWriter`	Writes text to a character-output stream, buffering characters to provide for the efficient writing of characters, arrays, and lines

Table 13-2 Selected classes used for input and output

Writing to a File

A. Construct stream objects using `BufferedOutputStream` and `OutputStream`
B. Create a writeable file by using the `Path` class method `newOutputStream(),which`
   - Creates a file if it does not already exist, if exist Opens the file for writing
   - and returns an `OutputStream` that can be used to write `bytes` to the file.
   - Table 13-4 shows the `StandardOpenOption` arguments useed with the `newOutputStream()` method

Writing to a File (cont'd.)

StandardOpenOption	Description
WRITE	Opens the file for writing
APPEND	Appends new data to the end of the file; use this option with WRITE or CREATE
TRUNCATE_EXISTING	Truncates the existing file to 0 bytes so the file contents are replaced; use this option with the WRITE option
CREATE_NEW	Creates a new file only if it does not exist; throws an exception if the file already exists
CREATE	Opens the file if it exists or creates a new file if it does not
DELETE_ON_CLOSE	Deletes the file when the stream is closed; used most often for temporary files that exist only for the duration of the program

Table 13-4 Selected StandardOpenOption constants

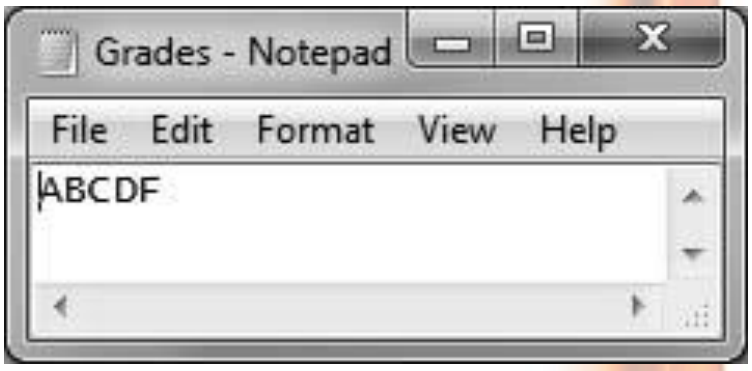
Note : specifying no option is the same as specifying both CREATE and TRUNCATE_EXISTING.

Selected OutputStream methods

OutputStream Method	Description
`void close()`	Closes the output stream and releases any system resources associated with the stream
`void flush()`	Flushes the output stream; if any bytes are buffered, they will be written
`void write(byte[] b)`	Writes all the bytes to the output stream from the specified `byte` array
`void write(byte[] b, int off, int len)`	Writes bytes to the output stream from the specified `byte` array starting at offset position `off` for a length of `len` characters

Table 13-3 Selected `OutputStream` methods

```
import java.io.*;
import static java.nio.file.StandardOpenOption.*;
public class FileOut
{
public static void main(String[] args)
{
Path file = Paths.get("d:\\Java-nio\\Grades.txt");
String s = "ABCDF";
byte[] data = s.getBytes();
try
{
OutputStream output = file.newOutputStream(CREATE);
BufferedOutputStream outf=new BufferedOutputStream (output);
outf.write(data);
outf.flush();
outf.close();
}
catch(Exception e)
{
System.out.println("Message: " + e);
}
}
}
```



```
import java.nio.file.*;
import java.io.*;
import static java.nio.file.StandardOpenOption.*;
public class FileOut
{
   public static void main(String[] args)
   {
      Path file =
         Paths.get("C:\\Java\\Chapter.13\\Grades.txt");
      String s = "ABCDF";
      byte[] data = s.getBytes();
      OutputStream output = null;
      try
      {
        output = new
           BufferedOutputStream(file.newOutputStream(CREATE));
        output.write(data);
        output.flush();
        output.close();
      }
      catch(Exception e)
      {
        System.out.println("Message: " + e);
      }
   }
}
```

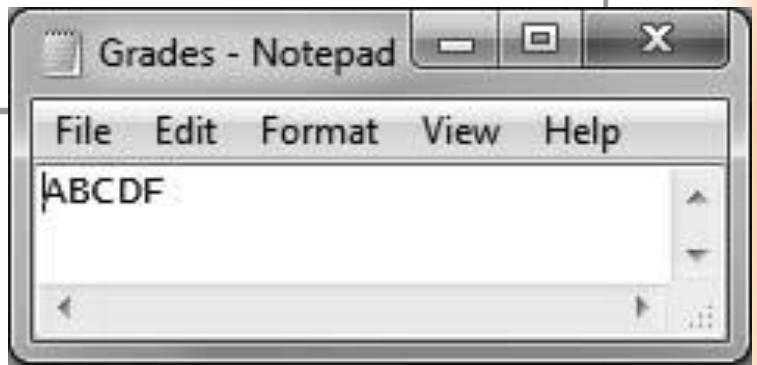


Figure 13-17 The FileOut class

Reading from a File

- Use an `InputStream` like you use an `OutputStream`
- Open a file for reading using the `Path` class method `newInputStream()`
  - The Method returns a stream that can read **bytes** from a file.

Reading a byte from a File

```
import java.nio.file.*;
import java.io.*;
public class ReadFile2
{
public static void main(String[] args)
{
Path file = Paths.get("D:\\Java\\Grades.txt");
InputStream input = null;
int b;
try
{
input = new BufferedInputStream (file.newInputStream());
b=input.read();
System.out.println(b);
input.close();
}
catch (IOException e)
{
System.out.println(e);
}
}
}
```

BufferedReader Methods

BufferedReader Method	Description
`close()`	Closes the stream and any resources associated with it
`read()`	Reads a single character
`read(char[] buffer, int off, int len)`	Reads characters into a portion of an array from position `off` for `len` characters
`readLine()`	Reads a line of text
`skip(long n)`	Skips the specified number of characters

Table 13-5 Selected `BufferedReader` methods

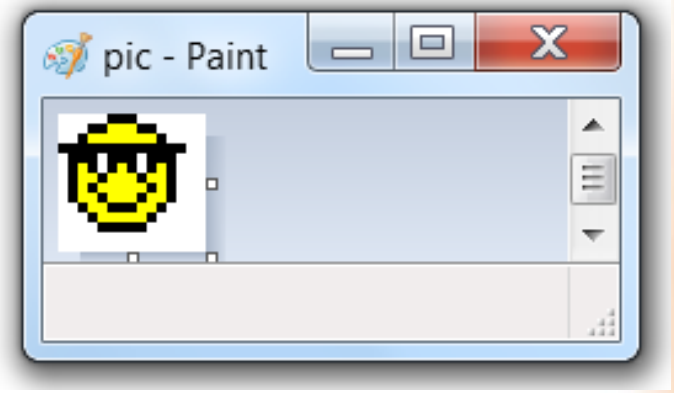
```
import java.nio.file.*;
import java.io.*;
public class ReadFile
{
   public static void main(String[] args)
   {
      Path file = Paths.get("C:\\Java\\Chapter.13\\Grades.txt");
      InputStream input = null;
      try
      {
         input = file.newInputStream();
         BufferedReader reader = new
           BufferedReader(new InputStreamReader(input));
         String s = null;
         s = reader.readLine();
         System.out.println(s);
         input.close();
      }
      catch (IOException e)
      {
         System.out.println(e);
      }
   }
}
```

Figure 13-19 The ReadFile class

Class java.io.InputStreamReader

- An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and translates them into characters according to a specified character encoding. The encoding that it uses may be specified by name, or the platform's default encoding may be accepted.

```
import java.nio.file.*;
import java.io.*;
import static java.nio.file.StandardOpenOption.*;
public class ByteWriter2 {
public static void main(String[] arguments) {
        int[] data = { 71, 73, 70, 56, 57, 97, 13, 0, 12, 0, 145, 0,  0, 255, 255, 255, 255,
            255, 0, 0, 0, 0, 0, 0, 0, 44, 0,  0, 0, 0, 13, 0, 12, 0, 0, 2, 38, 132, 45, 121, 11, 25,
            175, 150, 120, 20, 162, 132, 51, 110, 106, 239, 22, 8,  160, 56, 137, 96, 72, 77,
            33, 130, 86, 37, 219, 182, 230,  137, 89, 82, 181, 50, 220, 103, 20, 0, 59 };
       Path file = Paths.get("pic2.gif");
       OutputStream output = null;
        try {
               output = new BufferedOutputStream(file.newOutputStream(CREATE));
               for (int i = 0; i < data.length; i++)
               output.write(data[i]);
               output.close();
        } catch (IOException e) {
        System.out.println("error" + e.toString());
        }
   }
}
```



File Organization, Streams, and Buffers (cont'd.)

- In a Java application
  - **open** a file by creating an object and associating a stream of bytes with it.
  - When finish reading/writing, you should **Close** the file**.**
- When you leave a file open for no reason
  - you use computer resources and your computer's performance suffers , e.g: in a network, another program might be waiting to use the file.
  - the data might become inaccessible for a file to which you are writing data.