Unit IX

File Handling

Tutor: Pema Galey

Learning Outcomes

In this session, you will learn about:

- File Handling
- InputStream and OutputStream
- Reader and Writer

Using the File Class

- The File class in the `java.io` package provides various methods to access the properties of a file or a directory, such as file permissions, date and time of creation, and path of a directory.
- Constructors to create an instance of the File class are:
  - `File(File dirObj, String filename)`: Creates a new instance of the File class. The dirObj argument is a File object that specifies a directory. The filename argument specifies the name of the file.
  - `File(String directoryPath)`: Creates a new instance of the File class. The directoryPath argument specifies the path of the file.
  - `File(String directoryPath, String filename):` Creates a new instance of the File class. The argument directoryPath specifies the path of the file, and the filename argument specifies the name of the file.

Using the File Class

Methods of the File class are:

- `String getName()`:Retrieves the name of the specified file.
- `String getParent()`:Retrieves the name of the parent directory.
- `String getPath()`:Retrieves the path of the specified file.
- `String[] list()`:Displays the list of files and directories inside the specified directory.

Accessing File Properties

Accessing file properties:

- **`boolean delete()`**: The delete() method is used to delete a specified file. The delete() method returns true, if it successfully deletes the file otherwise false.
- **`boolean renameTo(File newName)`**: The renameTo() method is used to rename a specified file. The renameTo() method returns true, if it successfully renames the file otherwise false.

Random File Access

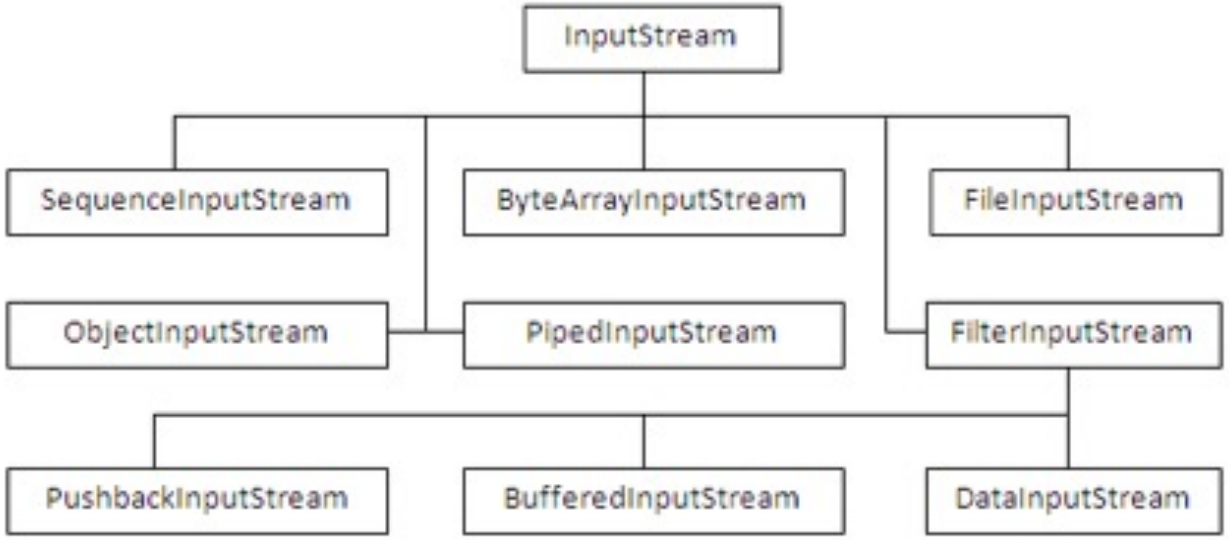
- You access the random files in Java by using the `RandomAccessFile` class. The constructor throws `FileNotFoundException`, if the `RandomAccessFile` class is unable to retrieve the name of the file to be created.
- Constructors to create an instance of the RandomAccessFile class are:
  - `RandomAccessFile(File fileObj, String mode):` Creates an instance of the random access file.
  - `RandomAccessFile(String name, String mode)`: Creates an instance of the random access file.
- Methods of the RandomAccessFile class are:
  - `void close():` Closes the random access file and releases the system resources, such as streams and file pointers associated with the file.

Random File Access

- `long getFilePointer()`: Retrieves the current position of the file pointer in the specified file.
- `int skipBytes(int n)`: Ignores the number of bytes from a file specified by the n argument.
- `long length()`: Retrieves the length of the specified file.
- `void seek(long position)`: Sets the current location of the file pointer at the specified position.

File Access using InputStream Class

- Input streams are the byte streams that read data in the form of bytes. The `InputStream` class is an abstract class that enables its subclasses to read data from different sources, such as a file and keyboard and displays the data on the monitor.
- The class hierarchy of the `InputStream` class.



File Access using InputStream Class

Methods of the `InputStream` class are:

- **`int read():`** Reads the next byte of data from an input stream. It returns –1 if it encounters the end of a stream.
- **`int read(byte b[]):`** Reads the number of bytes from the array specified by the b[] argument. The read() method returns –1 when the end of the file is encountered.
- **`int read(byte b[], int offset, int length):`** Reads the number of bytes from the array specified by the b[] argument. The argument offset specifies the starting position for the read operation, and length specifies the number of bytes to be read.
- **`available():`** Returns the total number of bytes available for reading in a stream.

File Access using InputStream Class

- `long skip(long n)`: Ignores the specified number of bytes from an input stream.
- `mark(int nbyte):` Places a mark at the current point in the input stream and this mark remains until the specified data bytes are read.
- `reset():` Places the file pointer to the previously set mark or at the beginning of the stream.
- `void close():` Releases the resources, such as streams and file pointers, associated with the file.
- `System.in` object:
  - The System class in the **java.lang** package has a static member variable, in that refers to the keyboard. The in variable is an instance of the InputStream class and is used to read data from the keyboard.

File Access using InputStream Class

The `FileInputStream` class:

- The `FileInputStream` class performs file input operations, such as reading data from a file.
- Constructors to create an instance of the `FileInputStream` class are:
  - `FileInputStream(File f):` Creates a file input stream that connects to an existing file to be used as data source.
  - `FileInputStream(String s):` Creates a file input stream that connects to an existing file to be used as data source.
  - `FileInputStream(FileDescriptor fdobj):` Creates a file input stream that connects to an existing file to be used as data source.

File Access using InputStream Class

Methods of the `FileInputStream` class are:

- `public int read()`: Reads a byte of data from the input stream.
- `public int read(byte b[], int offset, int length)`: Reads the specified number of bytes of data from the specified byte array.
- `public long skip(long n)`: Ignores the specified number of bytes of data from an input stream.

The `BufferedInputStream` class:

- The BufferedInputStream class accepts data from other input streams and stores it in a memory buffer.
- `BufferedInputStream(InputStream is)`: Creates an input stream and adds a buffer to it. The default size of the buffer is 512 bytes.
- `BufferedInputStream(InputStream is, int size)`: Creates an input stream and adds a buffer of size specified by the size argument.

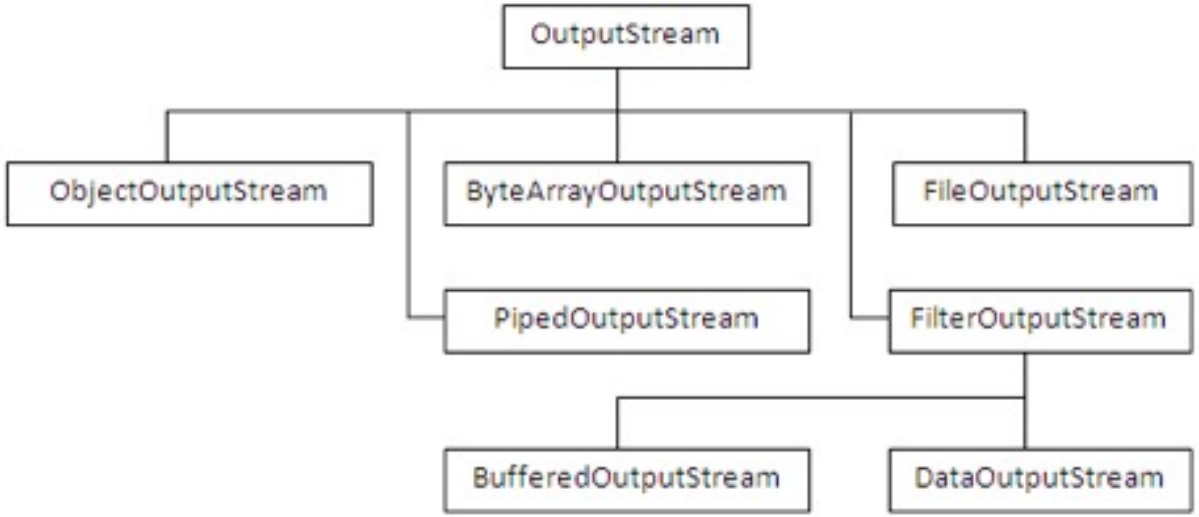
File Access using InputStream Class

- The `DataInputStream` class:
  - The `DataInputStream` class is used to read primitive data types, such as int, float, and double from an input stream.

- Methods of the `DataInputStream` class are:
  - `public final int read(byte b[]):` Retrieves a byte of input from the input stream.
  - `public final int read(byte b[], int offset, int length):` Reads the specified number of bytes of data from the specified byte array. The offset argument specifies the starting position for reading the data.

File Access using OutputStream Class

- Output streams are byte streams that write data in the form of bytes. The `OutputStream` class is an abstract class that enables its subclasses to write data to a file or screen.
- The class hierarchy of the `OutputStream` class.



File Access using OutputStream Class

- Methods in the `OutputStream` class:
  - `write(int b)`: Writes the specified byte to a file.
  - `write(byte b[])`: Writes an array of bytes specified by the b argument to a file.
  - `write(byte b[], int offset, int length)`: Writes an array of bytes specified by the b argument to a file. The offset argument determines the starting position for the byte array. The length argument specifies the number of bytes to be written.
  - `close()`: Closes the byte output stream.
  - `flush()`: Clears the buffers by removing any buffered output written on the disk.

File Access using OutputStream Class

- `System.out` object:
  - The `System` class in the `java.lang` package has a static member variable, out that represents the computer monitor. The out variable is an instance of the `PrintStream` class.
- The `FileOutputStream` class:
  - The `FileOutputStream` class is used to perform file output operations, such as writing to a file.
  - The various constructors to create an instance of the `FileOutputStream` class are:
    - `FileOutputStream(File f):` Creates a file stream that connects to an existing file to be used as destination for data. The File object is used to represent the required data file.

File Access using OutputStream Class

- `FileOutputStream(String s)`: Creates a file stream that connects to an existing file to be used as destination for data. The string argument provides the complete path of the file in the file system.
- `FileOutputStream(File f, boolean b):` Creates a file stream that connects to an existing file described by the file object. The true value for the boolean argument specifies that the file is opened in append mode.

- The various methods of the `FileOutputStream` class are:
  - `public void write(int b):` Writes the specified byte b to the output file stream.
  - `public void write(byte b[], int offset, int length):` Writes the total number of bytes specified by the length argument of the array b to the output file stream, starting from the offset position.

File Access using OutputStream Class

- The `BufferedOutputStream` class:
  - The `BufferedOutputStream` class creates a buffer in the memory and attaches this buffer to the output stream.
  - The various constructors to create an instance of the `BufferedOutputStream` class are:
    - `BufferedOutputStream(OutputStream os)`: Creates an output stream and adds a buffer to it. The default size of the buffer is 512 bytes.
    - `BufferedOutputStream(OutputStream os, int buflen)`: Creates an output stream and adds a buffer of size specified by the buflen integer variable.

File Access using OutputStream Class

- The various methods of `BufferedOutputStream` class are:
  - `public void write(int b)`: Writes the specified byte to the buffered output stream.
  - `public void write(byte b[], int offset, int len):` Writes the number of bytes from the b[] into the buffered output stream. The argument offset specifies the starting position for the write operation and length specifies the number of bytes to be written.
  - `public void flush()`: Writes all the bytes of data that are present in the buffer to the destination output device, such as a file.
- The `DataOutputStream` class:
  - The `DataOutputStream` class is used for writing the primary data types, such as int, float, and boolean on to an output stream.

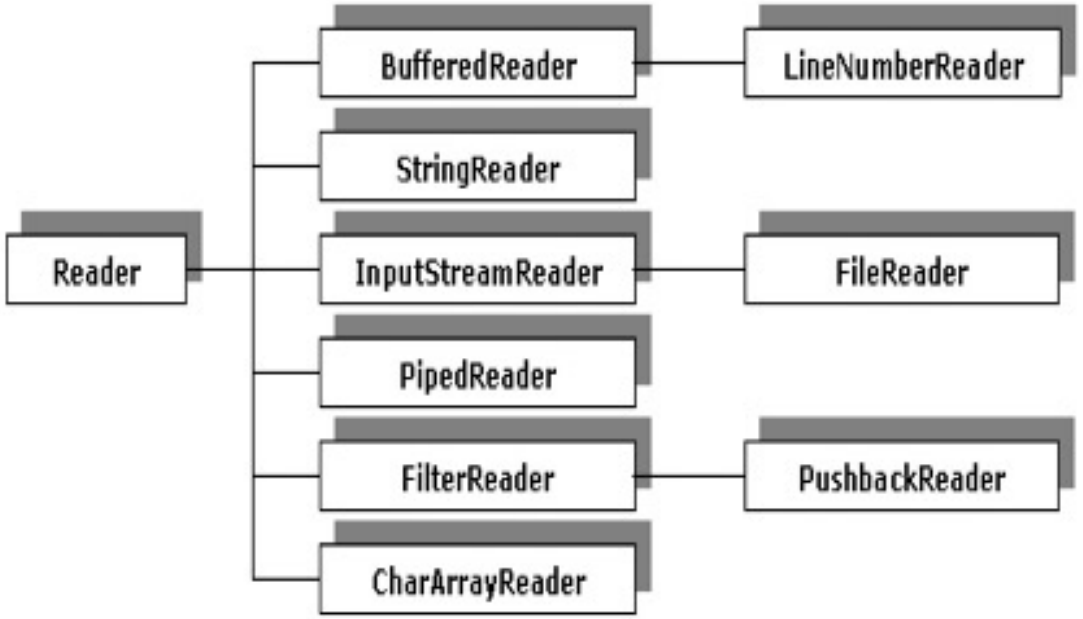
File Access using OutputStream Class

- The various methods of the `DataOutputStream` class are:
  - `public void write(byte b[], int offset, int length)`: Writes the number of bytes specified by length parameter from the buffer to the output stream starting from the position specified by offset parameter.
  - `public final void writeInt(int v):` Writes an integer value v to the output stream.
  - `public final void writeBytes(String s):` Writes a String s to the output stream.
  - `public final int size():` Returns the size of the output stream.
  - `public void flush() throws IOException`: Flushes the output stream.

- Demo:
  - File
  - RandomAccessFile
  - InputStream
  - OutputStream

Implementing Character Stream classes

- Using the `Reader` class:
  - The `java.io.Reader` class is an abstract class that provides various methods for reading the Unicode character data from input devices, such as hard disk, keyboard, and memory.
  - The following figure shows the Reader class hierarchy.



Implementing Character Stream classes

- The following tables lists the methods of the **`Reader`** class.

Method	Description
`int read()`	Reads a single character and returns an integer value of the character read. Returns `-1` if the end of file is encountered.
`int read(char buffer[])`	Reads character into the character array `buffer[]` and returns the actual number of characters that are successfully read. Returns `-1` if the end of file is encountered.
`abstract int read(char buffer[], int offset, int len)`	Reads the number of characters specified by the `len` parameter into `buffer` starting at the position specified by the `offset` parameter and returns the number of characters successfully read. Returns `-1` if the end of file is encountered.
`abstract void close()`	Closes the stream.
`void mark(int num)`	Places a mark at the current character in the stream that remains valid until the number of characters that are specified by the num parameter are read.
`void reset()`	Resets the pointer to the previously set mark.
`long skip(long num)`	Skips the number of characters specified by the `num` parameter and returns the number of characters that are skipped.

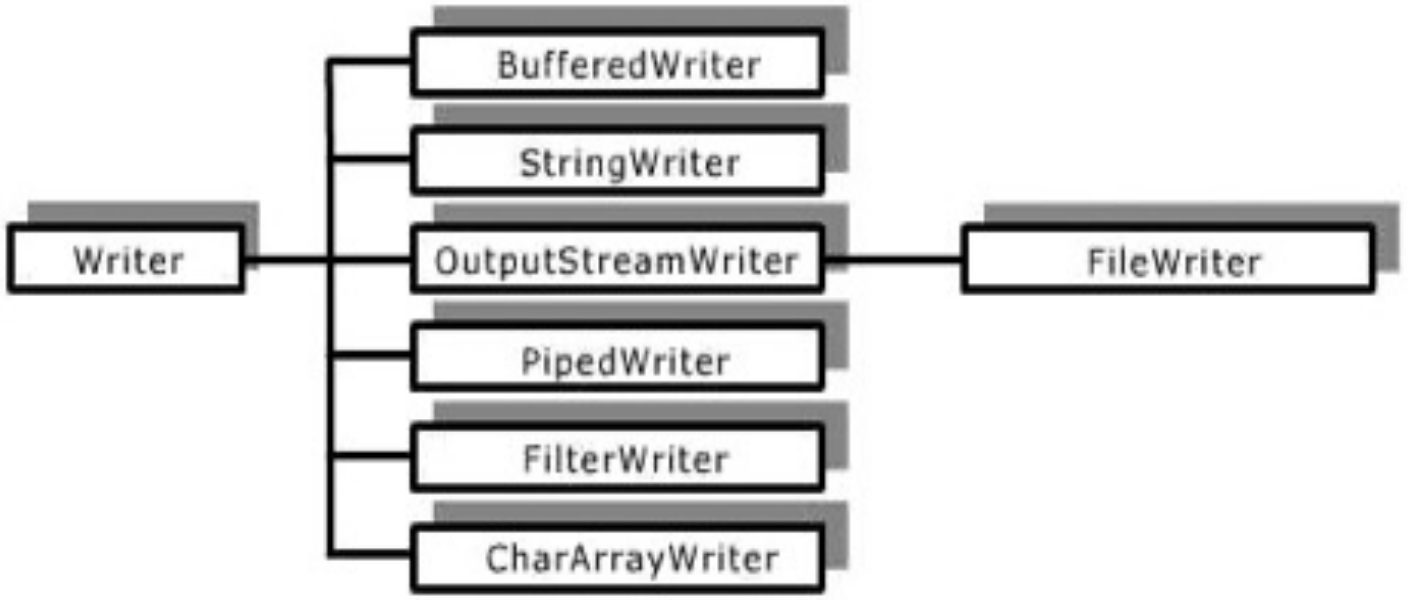
Implementing Character Stream classes

- The `FileReader` class:
  - The `FileReader` class is used for reading characters from a file, but it does not define any methods of its own.

- The `BufferedReader` class:
  - The `BufferedReader` stream reads text from a character input stream and attaches a buffer to the character input streams.

Implementing Character Stream classes

- Using the `Writer` class:
  - The `java.io.Writer` class is an abstract class that provides various methods for writing the Unicode character data to the output devices, such as hard disk and monitor.
  - The following figure shows the `Writer` class hierarchy.



Implementing Character Stream classes

- The following table lists the methods of the **`Writer`** class.

Method	***Description***
`void write(int ch)`	*Writes a character into the character-output stream. The written character is present in the 16 low-order bits of the given `ch` integer and the 16 high-order bits are ignored.*
`void write(char buffer[])`	*Writes a character array into the character-output stream.*
`abstract void write(char buffer[], int offset, int len)`	*Writes the number of characters specified by the `len` parameter from the character array starting at `buffer[offset]` to the character-output stream.*
`abstract void close()`	*Closes the character-output stream.*
`void write(String str)`	*Writes a string specified by the `str` parameter to the character-output stream.*
`void flush()`	*Flushes the output buffer.*

Implementing Character Stream classes

- The **`FileWriter`** class:
  - The following tables lists the constructors of the **`FileWriter`** class.

Constructor	***Description***
`FileWriter(String filename)`	*Creates an object of the `FileWriter` class where filename specifies the name of the object.*
`FileWriter(String filename, boolean appendData)`	*Creates an object of the `FileWriter` class. The `filename` parameter specifies the name of the object. The `appendData` parameter is a `boolean` variable that indicates whether to append data to the specified file.*
`FileWriter(File fileObj)`	*Creates a `FileWriter` class object where the `fileObj` parameter is an object of the `File` class.*

Implementing Character Stream classes

- The **`BufferedWriter`** class:
  - The **`BufferedWriter`** class extends the Writer class and writes text to a character output stream.
- The **`PrintWriter`** class:
  - The **`PrintWriter`** class extends the Writer class and writes the formatted character to the character output stream. This class defines various methods, such as **`print()`** and **`println()`** that provide formatted output.

Constructor	***Description***
`PrintWriter(OutputStream OStream)`	*Constructs an object of the `PrintWriter` class from the `OStream` object of the `OutputStream` class.*
`PrintWriter(OutputStream OStream, boolean autoflush)`	*Constructs an object of the `PrintWriter` class from the `Ostream` object of the `OutputStream` class. If the value of the `autoflush` parameter is `true` then the `println()` method flushes the output buffer.*
`PrintWriter(Writer outWriter, boolean autoflush)`	*Constructs an object of the `PrintWriter` class where `outWriter` is an object of the character output stream. If the value of the `autoflush` parameter is `true` then the `println()` method flushes the output buffer.*

Implementing Character Stream classes

- The following table lists the methods of the **`PrintWriter`** class.

Methods	***Description***
`void print(boolean b)`	*Prints a* `boolean` *value.*
`void print(int b)`	*Prints an integer value.*
`void print(char[] b)`	*Prints an array of characters.*
`void print(Object obj)`	*Prints an object.*
`void println()`	*Terminates the current line by placing the line separator string.*
`void println(boolean b)`	*Prints a* `boolean` *value and then terminates the current line.*

Implementing Object Serialization

- Serializing objects:
  - Serializing objects is the process of writing objects to a file. When an object is serialized, its state and other attributes are converted into an ordered series of bytes. This byte series can be written into streams.
- Interfaces and classes that support serialization are:
  - The `Serializable` interface
  - The `ObjectInputStream` class
  - The `ObjectOutputStream` class
- `Serializable` interface:
  - Is used for serializing objects in Java and does not define any method. An object that is saved and restored by the serialization process must implement the `Serializable` interface.

Implementing Object Serialization

- Using the **`ObjectOutputStream`** class for I/O operations:
  - The **`ObjectOutputStream`** class extends the **`OutputStream`** class and implements the **`ObjectOutput`** interface.
  - The following table lists the methods of the **`ObjectOutputStream`** class.

Methods	***Description***
`public void writeObject(Object obj) throws IOException`	*Writes the `obj` object to the `ObjectOutputstream` stream.*
`public void flush() throws IOException`	*Ensures that the data stored in the buffer is written to a file to which the stream is connected.*
`public void close() throws IOException`	*Closes the stream and releases all the resources occupied by the stream.*

Implementing Object Serialization

- The **`transient`** keyword: The data members of a class that are not required to be serialized are declared as transient. The serialization process ignores the transient variables.
- Using the **`ObjectInputStream`** class for I/O operations:
  - The **`ObjectInputStream`** class extends the **`InputStream`** class and implements the **`ObjectInput`** interface.
  - The following table lists the various methods of the **`ObjectInputStream`** class.

Methods	***Description***
`public final Object readObject() throws IOException, ClassNotFoundException`	*Reads an object from* `ObjectInputStream`.
`public void close() throws IOException`	*Closes a stream and releases the resources occupied by the stream.*
`public String readLine() throws IOException`	*Reads a line that has been terminated by End Of File (EOF) or the new line character (\n).*

Implementing Object Serialization

- A class implements the `Serializable` interface in order to serialize its objects. The class that implements the `Serializable` interface may be a user-defined class.

Demo

- Reader and Writer

Home Work

Problem Statement:

- Global Systems, Inc. is a company dealing with selling and buying of computer products. Steve wants to create a Java application that stores the sales details of the computer products in a text file. Help Steve create the Java application. The sales details should include the following information:
  - Product Id
  - Product Name
  - Price
  - Date

Thank you!