Java

I/O

I/O Part-1

File

- Long-term storage of large amounts of data
- Persistent data exists after termination of program
- Files stored on secondary storage devices
    - Magnetic disks
    - Optical disks
    - Magnetic tapes
- Sequential and random access files

File Class

- Provides useful information about a file or directory
- Does not open files or process files
- To obtain or manipulate path, time, date, permissions etc
- Constructor
  - File(String directoryPath)
  - File(String directoryPath, String fileName)
  - File(File dirObj, String fileName)
- Example: **FileDemo.java**

Directory Class

- Directories are also files
- Contains list of files and directories
- For Directory isDirectory() returns true.
- *String[] list()*
  - returns an array of strings that gives the files and directories contained
- *File[] listFiles()*
  - Returns array of File objects
- Example: **DirectoryDemo.java**

Stream Classes

- Java views a File as a stream of bytes.
  - File ends with end-of-file marker or a specific byte number.
  - File as a stream of bytes associated with an object.
  - Java also associates streams with devices
    - *System.in*, *System.out*, and *System.err*
  - Streams can be redirected
- Stream is an abstraction that either produces or consumes information.

Stream Classes

- Java's stream-based I/O is built upon four abstract classes.
  - InputStream, OutputStream (for byte streams)
  - Reader , Writer (for character streams)
- They form separate hierarchies.
- You should use the character stream classes when working with characters or strings and use the byte stream classes when working with bytes or other binary objects

Byte-Stream Classes Hierarchy

java.lang.Object
File
FileDescriptor
InputStream
ByteArrayInputStream
FileInputStream
FilterInputStream
BufferedInputStream
DataInputStream
PushbackInputStream
ObjectInputStream
PipedInputStream
SequenceInputStream
OutputStream
ByteArrayOutputStream
FileOutputStream
FilterOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
ObjectOutputStream
PipedOutputStream

Character-Stream Classes Hierarchy

RandomAccessFile
Reader
BufferedReader
LineNumberReader
CharArrayReader
FilterReader
PushbackReader
InputStreamReader
FileReader
PipedReader
StringReader
Writer
BufferedWriter
CharArrayWriter
FilterWriter
OutputStreamWriter
FileWriter
PipedWriter
PrintWriter
StringWriter

Byte Stream Classes

- Byte-Stream classes are topped by InputStream and OutputStream classes.
- InputStream is an abstract class that defines Java's model of streaming byte input.

  int available() *void close()* *int read()*

  int read(byte buff[]) *int read(byte buff[], int off, int num)*
- OutputStream is an abstract class that defines Java's model of streaming byte output.

  void flush() *void close()* *void write(int b)*

  void write(byte buff[]) *void write(byte buff[], int off, int num)*

FileInputStream

- FileInputStream class creates an InputStream that you can use to read bytes from a file.
- Constructors
  - FileInputStream(String filePath)
  - FileInputStream(File fileObj)
- Example: **FileInputStreamDemo.java**

FileOutputStream

- FileInputStream class creates an OutputStream that you can use to write bytes to a file.
- Constructors
  - FileOutputStream(String filePath)
  - FileOutputStream(File fileObj)
  - FileOutputStream(String path, boolean append)
  - FileOutputStream(File obj, boolean append)
- Example:

  FileOutputStreamDemo.java, FileCopyDemo.java

DataInputStream & DataOutputStream

- DataInputStream & DataOutputStream enable to write or read primitive data to or from a stream.
- They implement the DataOutput & DataInput interfaces respectively.
- Constructors
  - DataOutputStream(OutputStream os)
  - DataInputStream(InputStream is)
- Example: **DataIODemo.java**

I/O Part-2

Part-1

- File & Directory
- Java Stream Classes
  - Byte Streams
  - Character Streams
- ByteStream
  - InputStream
    - FileInputStream
    - DataInputStream
  - OutputStream
    - FileOutputStream
    - DataOutputStream

Character Streams

- Character Stream classes are topped by Reader and Writer class.
- Reader is an abstract class that defines Java's model of streaming character input.

  void close() *int read()*

  int read(char buff[]) *int read(char buff[], int off, int num)*
- Writer is an abstract class that defines Java's model of streaming character output.

  void flush() *void close()* *void write(int ch)*

  void write(char buff[]) *void write(char buff[], int off, int num)*

  void write(String s) *void write(String s, int off, int num)*

FileReader

- The FileReader class creates a Reader that you can use to read the contents of a file.
- Constructors
  - FileReader(String filePath)
  - FileReader(File fileObj)
- Example: **FileReaderDemo.java**

FileWriter

- The FileWriter class creates a Writer that you can use to write to a file.
- Constructors
  - FileWriter(String filePath)
  - FileWriter(File fileObj)
  - FileWriter(String path, boolean append)
  - FileWriter(File obj, boolean append)
- Example: **FileWriterDemo.java**

BufferedReader

- The BufferedReader is a Reader that buffers input.
- It improves performance by reducing the number of times data us actually physically read from the input stream.
- Constructors
  - BufferedReader(Reader reader)
  - BufferedReader(Reader reader, int buffSize)
- Example: **BufferedReaderDemo.java**

BufferedWriter

- The BufferedWriter is a Writer that buffers output.
- It improves performance by reducing the number of times data us actually physically written to the output stream.
- Constructors
  - BufferedWriter(Writer writer)
  - BufferedWriter(Writer writer, int buffSize)
- Example: **BufferedWriterDemo.java**

Console

- Java SE 6 adds the Console class. It is used to read and write to the console.
- It supplies no constructor. A Console object is obtained by calling System.console(), which is shown here static Console console().
- Important Methods: printf, readLine, readPassword
- Example: **ConsoleDemo.java**

Serialization

- Serialization is the process of writing the state of an object to a byte stream.
- This is useful when you want to save the state of your program to a persistent storage such as file.
- Later these objects can be restored by using the process of deserialization.
- They work properly in the scenarios where one object contains a reference to another object or to the same object or more complex one.
- Serialization can be achieved by implementing Serializable interface.

ObjectInputStream & ObjectOutputStream

- The ObjectInputStream class extends the InputStream class.
- It is responsible for reading objects from a stream.
- The ObjectOutputStream class extends the OutputStream class.
- It is responsible for writing objects to a stream.
- Example: **ObjectSerializationDemo.java**

End