1. **Stream:** In java all I/O sources are handled by same abstraction called Stream.
   1. Stream is a logical entity that either consume or produce data.
   2. Through stream abstraction operation can be done consistently irrespective of different IO devices.
   3. Java.io consists library for IO.
2. **File**: File represents a directory or file on disc.
   1. **Constructor**: File(dir), File(dir, file), File(dirObj, file), File(uriObj)
   2. **Methods**: getName(), getPath(), getAbsolutePath(), exists(), isFile(), isDirectory(), canRead(), canWrite(), canExecute(), isAbsolute(), renameTo(fileObj), delete(), deleteOnExit(), getFreeSpace(), isHidden(), setReadOnly(), mkdir(), mkdirs()
      1. **Java8**: toPath(), list([FilenameFilter]), listFiles([FilenameFilter]), listFiles([FileFilter])
   3. Both ‘/’ and ‘\\’ is permissible while specifying a path.
   4. isFiles() returns false if called on device drivers or name pipes.
   5. delete() method deletes a directory if empty.
   6. String [] List(): No of file name returned by list can be controlled by pass it an instance of FilenameFiler: accept(File dir, String fileName)
   7. Mkdirs creats only last directory specified mkdirs() create all the directories in path if doesn’t exist.
3. **Autocloseable**: Beginning with JDK 7 Closeable interface extends Autocloseable.
4. **Flushable**: flush(): Provides functionality to write buffered data to be written on o/p device.
5. **ByteStreams**: InputStream and OutputStream designed for byte streams.
   1. **InputStream**: All methods throw IOException except mark() and markSupported()
      1. **Methods**: available(), close(), mark(numBytes), markSupported(), int read([buffer, offset, numBytes]) [read numBytes starting from offset into buffer], reset(), skip(numBytes), read()
      2. **FileInputStream**: FileInputStream([filePath]), FileInputStream([fileObj])
         1. FIS doesn’t support mark/reset
      3. **ByteArrayInputStream**: BAIS(buffer, [start, numBytes])
         1. It supports mark and reset.
         2. If reset is called without marking pointer would be at starting of the stream.
   2. **OutputStream**:
      1. **Methods**: close(), flush(), write(numbytes), write(buffer), write(buffer, offset, numBytes)
      2. **FileOutputStream**: FileOutputStream(filePath, [append]), FileOutputStream(fileObj, [append])
         1. If file is not present it creates one.
         2. If append is true file would be opened to append otherwise would be truncated.
         3. **ByteArrayOutputStream**: BAOS([numBytes])
            1. Default buffer size is 32 otherwise numBytes.
            2. Methods: reset(), writeTo()
6. **Buffered Stream**: Buffered streams attaches memory buffer to a IO stream. This allows more than one byte to be read/write on a stream. This also allows skipping, marking and resetting of stream.
   1. **BufferedInputStream**: BufferedInputStream **extends** FilterInputStream
      1. Constructor: BIS(is,[ bufSize])
   2. **BufferedOutputStream**:
      1. Constructor: BOS()
      2. Method: flush()
   3. **PushbackInputStream**: PIS is used for the byte to be read and returned to the IS again.
      1. Constructor: PIS(is, numBuf)
      2. Method: unread(num), unread(buf ), unread(buff, offset, numBytes)
   4. **SequenceInputStream**: SIS concatenates multiple input stream. Operationally it fulfills read request from 1st stream until it runs out and then switches to the 2nd stream and so on.
      1. Constructor: SIS(is1, is2)
   5. **PrintStream**: PS is provides output stream of System.out . **public** **class** PrintStream **extends** FilterOutputStream **implements** Appendable, Closeable
      1. Constructor: PS(os, [autoFlushOn, charSet]), PS(file/fileName, [charset/charset])
      2. Method: print(), println, printf([locale], fmtString, args), format([locale], fmtString, args)
   6. **DataInputStream**: Reads data from IS and coverts to primitive.
      1. Methods: readInt, readDouble, readBoolean
   7. **DataOutputStream**: convert primitive data to sequence of byte and writes to output stream.
      1. Methods: writeInt, writeDouble, writeBoolean
7. **RandomAccessFile**: RAF encapsulates random access to a file. RandomAccessFile **implements** DataOutput, DataInput, Closeable
   1. **Constructor**: RAF(file, access)
   2. **Access**:
      1. "r": Open for reading only. Invoking any of the write methods of the resulting object will cause an IOException to be thrown.
      2. "rw": Open for reading and writing. If the file does not already exist then an attempt will be made to create it.
      3. "rws": Open for reading and writing, as with "rw", and also require that every update to the file's content or metadata be written synchronously to the underlying storage device.
      4. "rwd": Open for reading and writing, as with "rw", and also require that every update to the file's content be written synchronously to the underlying storage device.
   3. **Methods**: setLength() [shortens or extend file length], seek(newPos)
8. **Character Streams**: Character streams are used to work with characters.
   1. **Reader**:
      1. **Methods**: close, mark(numChars), markSupported(), read([buffer]), ready(), reset, skip(numChars)
      2. **FileReader**:
         1. Constructor: FR(filePath/fileObj)
      3. **CharArrayReader**:
         1. Constructor: CAR(array, [start, numChars])
      4. **BufferedReader**:
         1. Constructor: BR(is, [buffer])
         2. Methods: lines() [java 8], readLine([ignoreLF])
      5. **PushbackReader**:
         1. Constructor: PR(is, [bufSize])
         2. Methods: unread(ch/buffer, [offset, numChars])
   2. **Writer**:
      1. **Methods**: append(ch/ch\_seq), write(ch/buffer/str, [offset, numChars])
      2. **FileWriter**:
         1. Constructor: FW(filePath/fileObj, [append])
      3. **CharArraywriter**:
         1. Constructor: CAW()
         2. Methods: toCharArray()
      4. **BufferedWriter**:
         1. Constructor: BR(is, [bufSize])
            1. Default buffer size is 8192.
      5. **PrintWriter**:
         1. Constructor: PW(os/file/filePath, [autoflashon])
         2. Methods: print, println, printf, format

try (Stream<Path> paths = Files.walk(Paths.get("/home/you/Desktop"))) {

paths

.filter(Files::isRegularFile)

.forEach(System.out::println);

}