**COMPUTER FILES**

*( a collection of data stored on a non - volatile device)*

**TWO TYPES OF STORAGE DEVICE:**

1.VOLATILE

2.NON - VOLATILE

* **VOLATILE STORAGE**
* temporary
* lost when computer loses its power
* **NON - VOLATILE**
* permanent
* saved on a disk

**TWO CATEGORIES OF FILE:**

1.TEXT FILE

2.BINARY FILE

* **TEXT FILE**
* consists of data that can be read in a text editor.
* encoded using a scheme.
* Scheme - ASCII and UNICODE

**example**: program files and application files

* **BINARY FILE**
* contains data that is NOT encoded as TEXT
* files has contains binary format
* cannot be understood by viewing them in text editor

**example**: images, music and .class

* **common operations done with the file:**

*Determining whether and where a path pr file exists*

* Opening a file
* Reading from a file
* Writing to a file
* Closing a file
* Deleting a file
* **common characteristics of a text file and binary file are**
* size
* name and
* date and time of creation
* **MAIN DIRECTORY / ROOT DIRECTORY**
* where permanent files are commonly stored
* **FOLDERS / DIRECTORIES**
* are used to stored files
* create folders within folders
* **PATH**
* it is a complete list of of the disk drive plus the hierarchy of directory/directories in which the file is located

**example:** path: C:\Java\Chapter8\example.txt

* **\ path delimiter (backslash)**
* special character used to separate path components.

**THE PATH AND FILES CLASSES**

* **PATH CLASS**
* creates objects that contain information about files and directories such as:
* sizes
* locations
* creation dates
* and is used to check whether a file or directory exists.
* **FILES CLASSES**
* performs operations on files and directories such as determining their attributes, creating input and output streams, and deleting them.
* **to use both the Path and Files classes, insert:**
* import java.nio.file.\*;
* **to create a path:**
* Path filePath =

Paths.get("C:\\Java\\Chapter8\\sample.txt")

**TWO TYPES OF PATH**

1.ABSOLUTE

2.RELATIVE

* **ABSOLUTE PATH**
* a complete path
* does not require any other information to locate a file on a system

**example:** C:\Java\Chapter8\sample.txt

* **RELATIVE PATH**
* depends on other path information

**examples:** sample

chapter8\sample.txt

Java\chapter8

* **toAbsolutePath( )**
* converts a relative path to an absolute path

**PATH CLASS METHOD**

**METHOD**

1.toString( )

2.getFileName( )

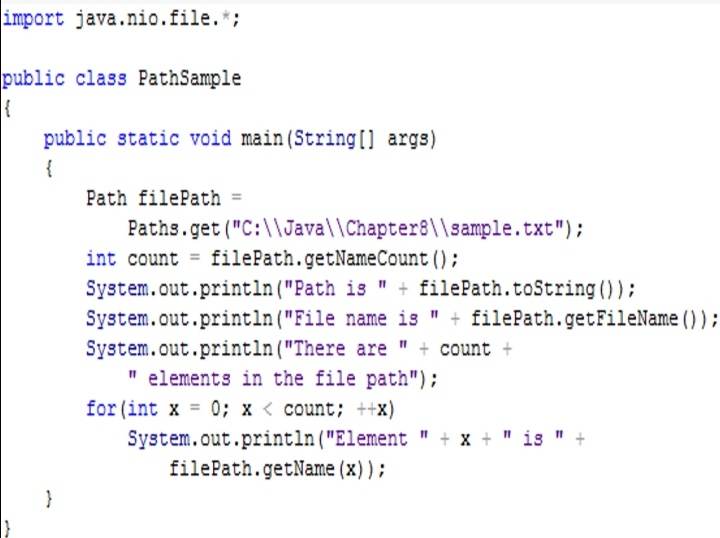
3.getNameCount( )

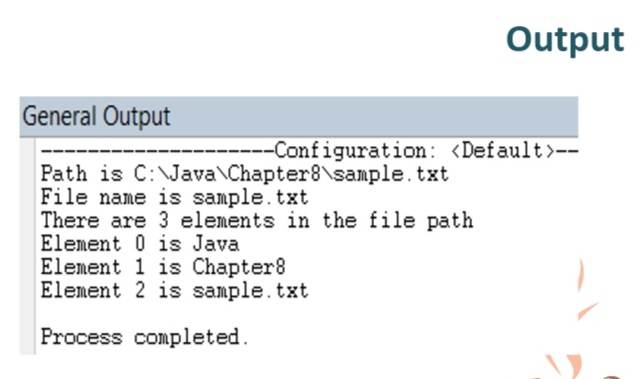
4.getName( )

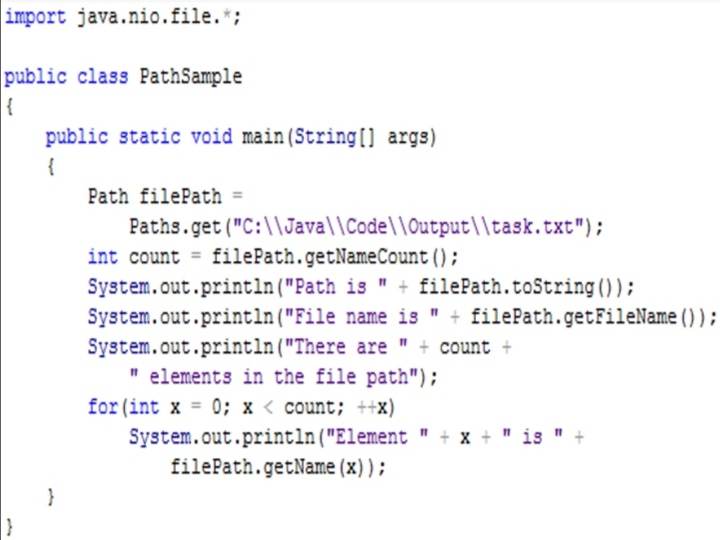
* **String toString( )**
* returns the String representation of tge *Path*, eliminating double backslashes
* **Path getFile Name( )**

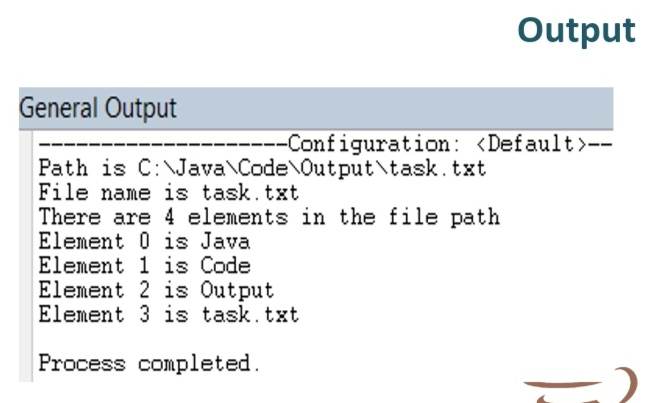
returns the file or directory denoted by this *Path*; this is the last item in the sequence of the name elements

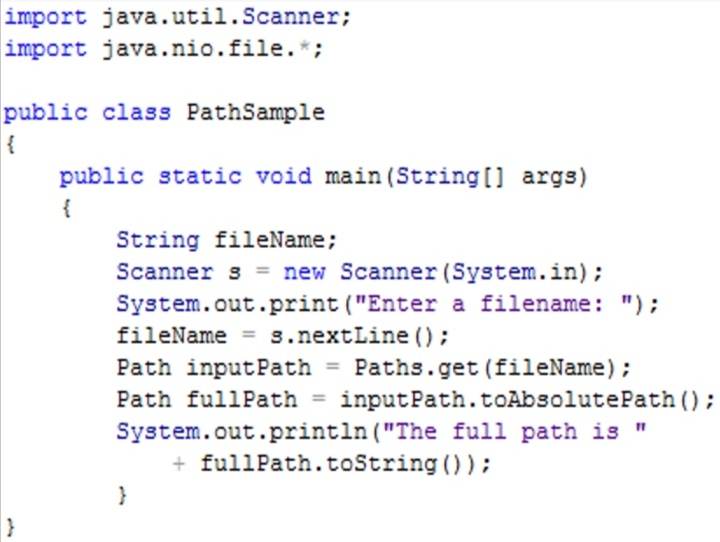
* **int getNameCount( )**
* returns the number of name elements in the *Path*
* **Path getName( )**
* returns the name in the position of the *Path* specified by the integer parameter

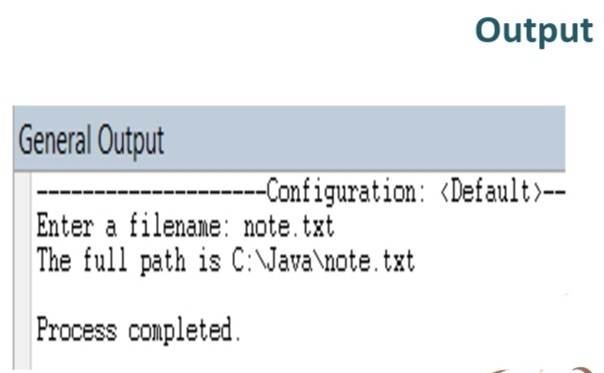












**File Accessibility**

* **checkAccess( )**
* checks if a file exists and if a program can access it
* **import statement:**
* import static java.nio.file.AccessMode.\*;
* **Syntax:**
* filePath.getFileSystem( ).provider( ).checkAccess( );

**ARGUMENTS of the checkAccess( ) Method** *(checkAccess( ) Arguments)*

**ARGUMENTS**

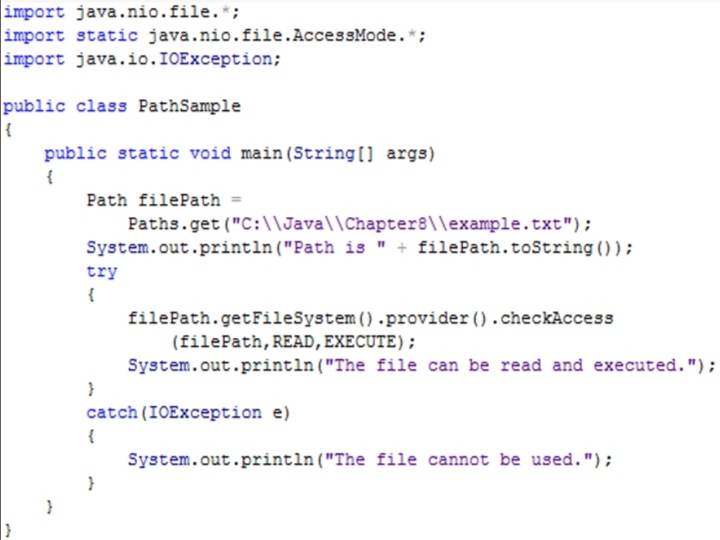
1.None

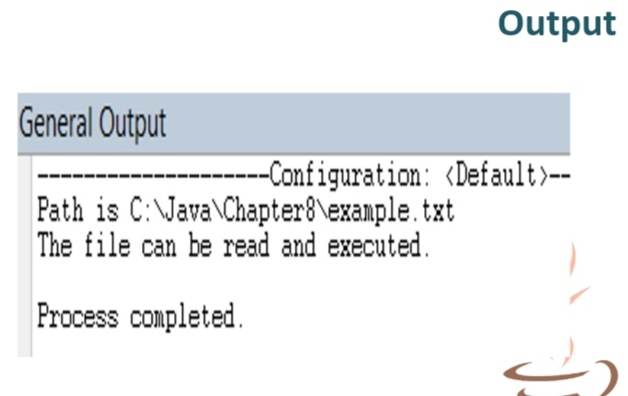
2.READ

3.WRITE

4.EXECUTE

* **None**
* checks whether the file exists
* **READ**
* checks whether the file exists and whether the program has permission to read the file
* **WRITE**
* checks whether the file exists and whether the program has permission to write to the file
* **EXECUTE**
* checks whether the file exists and whether the program has permission to execute the file





**delete( )**

* accepts a Path parameter and removes the last element (file or directory) in a path or throws an exception if the deletion is unsuccessful

**deleteIfExists( )**

* used to remove a file without encountering an exception if the file does not exists

**EXCEPTIONS**

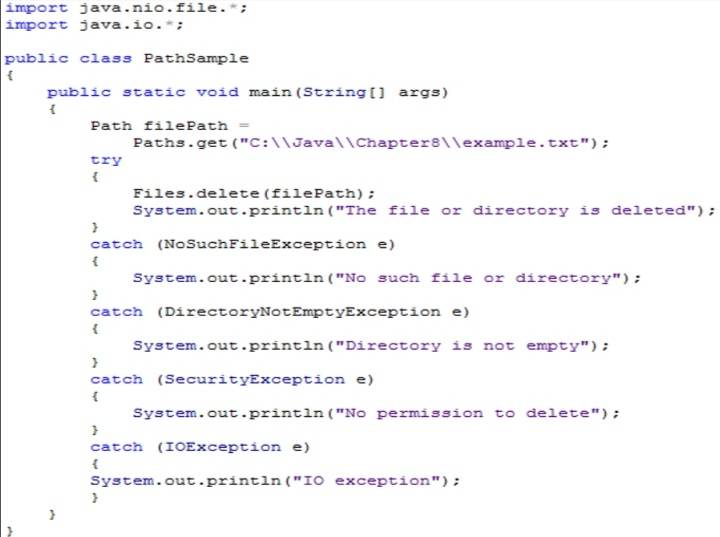
1.NoSuchFileException

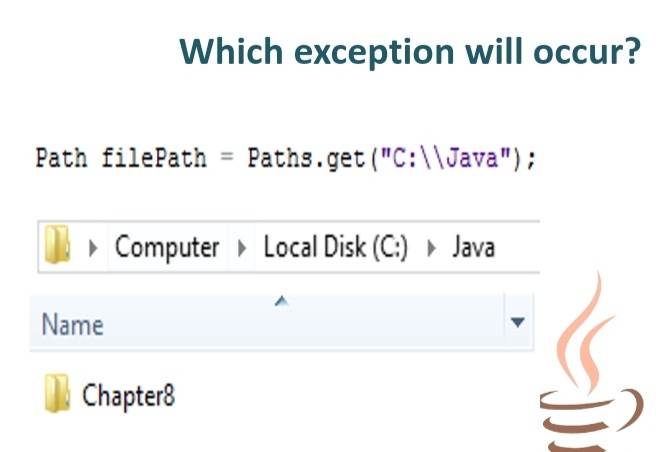
2.DirectoryNotEmptyException

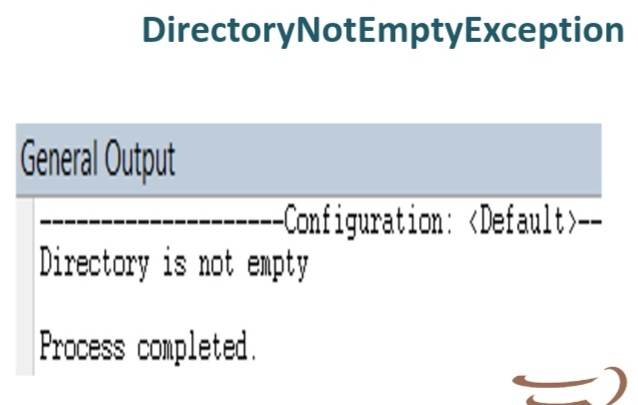
3.SecurityException

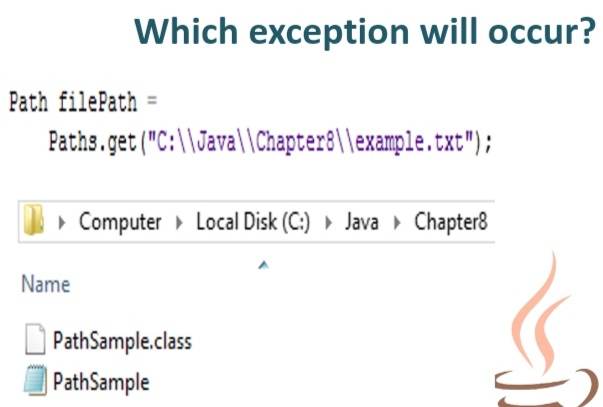
4.IOException

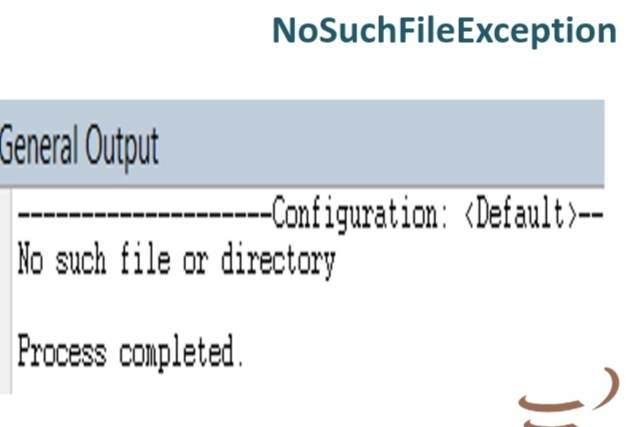
* **NoSuchFileException**
* was thrown if there is an attempt to delete a file that does not exist
* **DirectoryNotEmptyException**
* was thrown if there is an attempt to delete a directory that has files
* **SecurityException**
* was thrown if there is an attempt to delete a file without permission
* **IOException**
* other input errors











**RETRIEVING FILE ATTRIBUTES**

* **readAttributes( )**
* retrieves information about a file
* method that returns an instance of the BasicFileAttribute.class

**example**: BasicFileAttributes fileAtt =

Files.readAttributes(filePath, BasicFileAttributes.class);

* **import statement:**
* import java.nio.file.attribute.\*;
* **Sample Statement:**
* BasicFileAttributes fileAtt = Files.readAttributes(filePath, BasicFileAttributes.class);

**TWO ARGUMENTS OF readAttributes( )**

1.Path

2.BasicFileAttributes.class

* **PATH**
* it is a complete list of of the disk drive plus the hierarchy of directory/directories in which the file is located
* **Basic file attributes**
* attributes that are common to many file systems and consist of mandatory and optional file attributes as defined by this interface.

**BasicFileAttributes Methods**

METHODS

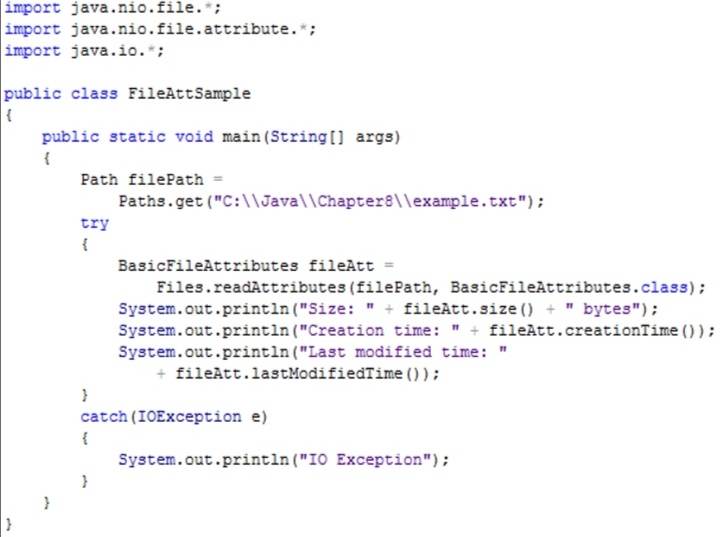
1.size( )

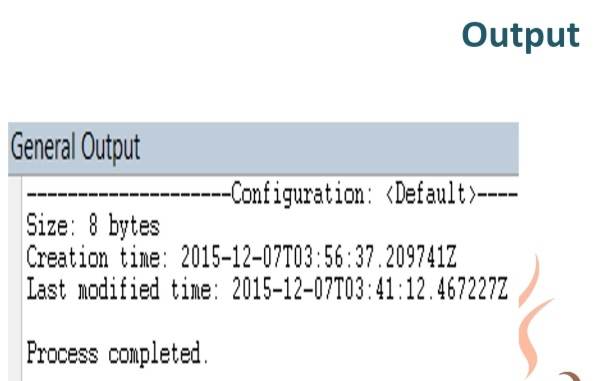
2.creationTime( )

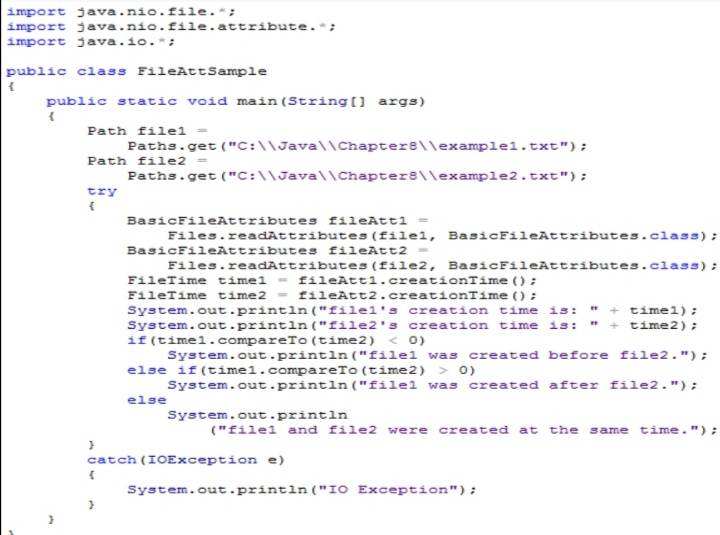
3.lastModifiedTime( )

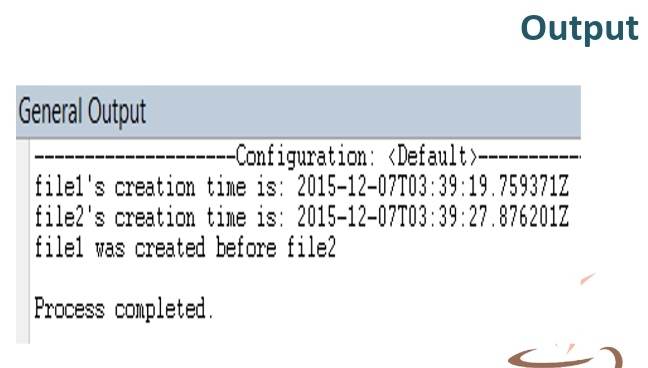
4.CompareTo( )

* **size( )**
* returns the size of the file in bytes
* **creationTime( )**
* returns the date and time the files was created
* **lastModifiedTime( )**
* returns the date and time the files was last edited
* **CompareTo( )**
* compares relationship between values retrieved from creationTime( ) or lastModifiedTime( )









**FILE ORGANIZATION, STREAMS, and BUFFERS**

**FILE ORGANIZATION**

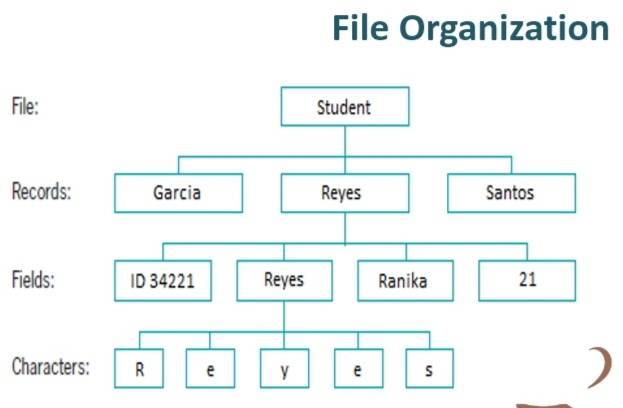
1.Character

2.Field

3.Record

4.File

* **Character**
* any letter, number, or other special symbol
* smallest useful piece of data
* **Field**
* group of characters that has some meaning
* **Record**
* collection of fields that contain data about an entity
* **File**
* group of related records



**SEQUENTIAL ACCESS FILE**

* a data file that is used when each record in a file is accessed one after another in the order in which it was stored

***a record’s fields can be organized into a single line or can be separated by a character.***

**COMMA - SEPARATED VALUES**

* refers to values im a record that are separated by commas

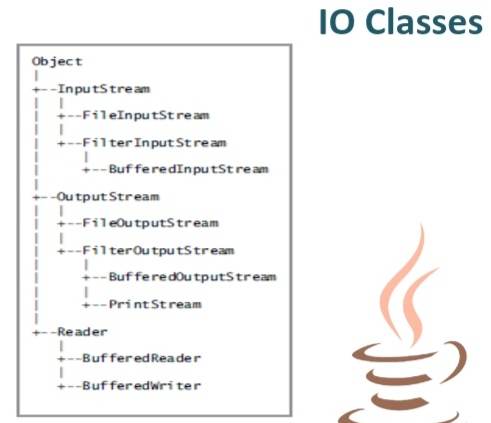
**STREAM** *( flow of data)*

1. input stream
2. output stream

* **Input Stream**
* the data is taken from a source (file or keyboard) and is delivered into your program
* **Output Stream**
* the data is delivered from your program to a destination (file or screen)

**BUFFER** *( a memory location into which you can write data)*

* **Flushing**
* clears any bytes that have been sent to a buffer fot output but have not yet been displayed to a hardware device.



**THE IO CLASSES**

**CLASS**

1. Inputstream
2. FileInputStream
3. BufferedInputStream
4. OutputStream
5. FileOutputStream
6. BufferedOutputStream
7. PrintStream
8. Reader
9. BufferedReader
10. BufferedWriter

* **InputStream**
* abstract class that contains methods for performing input
* **FileInputStream**
* provides the capability to read disk from files
* **BufferedInputStream**
* handles input from a system's standard or default input device

(usually the keyboard)

* **OutputStream**
* abstracts that contains methods for performing output
* **FileOutputStream**
* provides the capability to write to disk files
* **BufferedOutputStream**
* handles input from a system's standard or default output device

(usually the monitor)

* **PrintStream**
* contains methods for performing output that never throws an exception
* **Reader**
* abstract class for reading character streams
* **BufferedReader**
* reads text from a character-input stream
* **BufferedWriter**
* writes text to a character-output stream

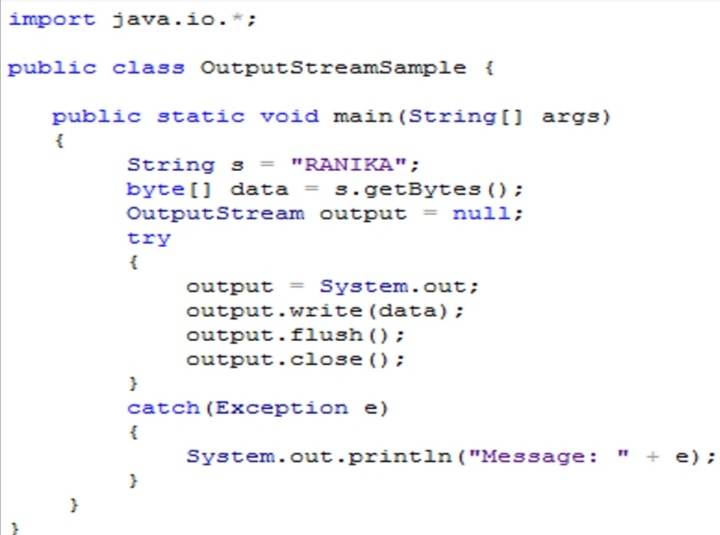
**COMMON METHOD OF THE OUTPUTSTREAM CLASS**

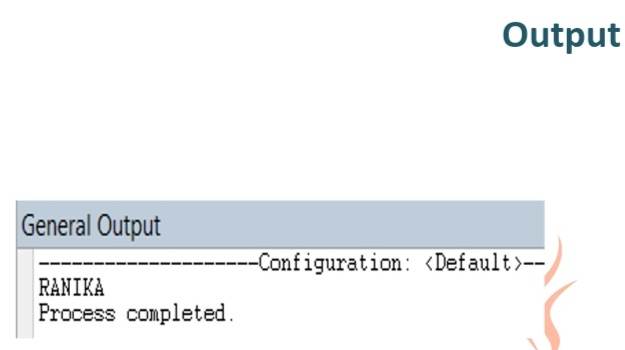
**OutputStream Methods**

METHOD

1. close( )
2. flush( )
3. write( byte[ ] b )
4. write( byte[ ] b, int off, int len)

* **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 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





**newOutputStream( )**

* is used to create a writable file. A path and a StandardOpenOption argument are passed to this method.
* This method creates a file if it hasn't existed yet, opens the file for writing and returns an OutputStream that can be used to write bytes to the file

**newInputStream( )**

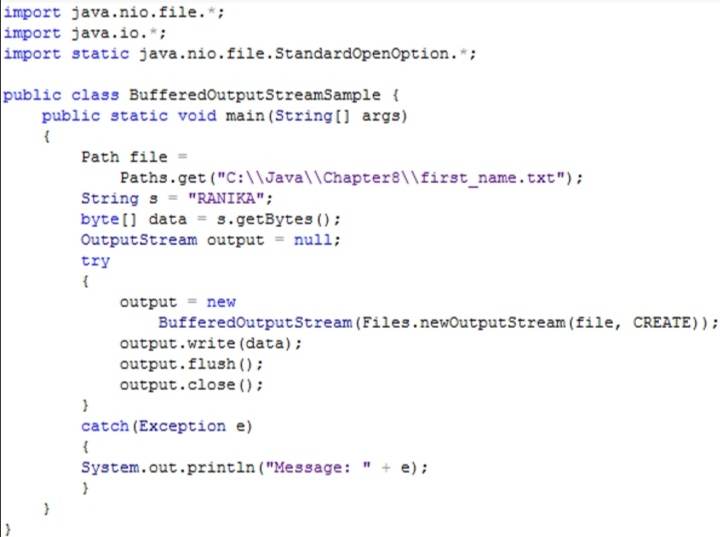
* is used to open a file for reading.
* This method accepts a path parameter and returns a stream that can read bytes from a file.

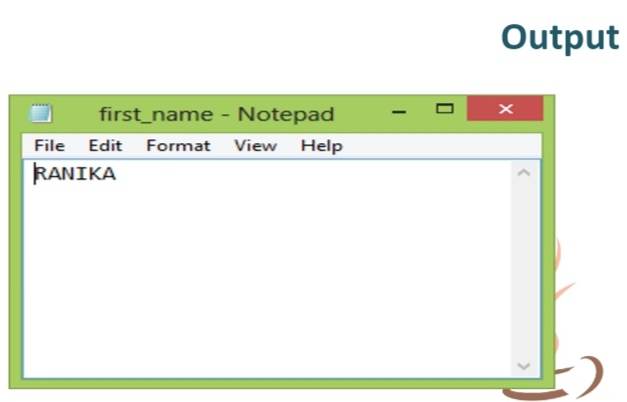
**StandardOpenOption Arguments**

**StandardOpenOption**

1. WRITE
2. APPEND
3. TRUNCATE\_EXISTING
4. CREATE\_NEW
5. CREATE
6. DELETE\_ON\_CLOSE

* **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 byte so the file contents are replaced, use this option with the WRITE option
* **CREATE\_NEW**
* creates a new file only if it hasn't existed yet
* **CREATE**
* opens the file if it exists or creates a new file if it hasn't existed yet
* **DELETE\_ON\_CLOSE**
* deletes the file when the stream is closed





**BufferedReader Methods *(****object is declared to read a line of text from a character-input stream, buffering characters so reading is more efficient)*

**METHOD**

1. close( )
2. read( )
3. read char [ ] buffer, int off, int len)
4. readLine( )
5. skip(long n)

* **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 fof len characters
* **readLine( )**
* reads a line of text
* **skip(long n)**
* skips the specified number of characters

**BufferedWriter Methods** *(class writes text to an output stream, buffering the character)*

**METHOD**

1. close( )
2. flush( )
3. newline( )
4. write(String s, int off, int len)
5. write(char [ ] array, int off, int len)
6. write(int c)

* **close( )**
* closes the stream, flushing it first
* **flush( )**
* flushes the stream
* **newline( )**
* writes a line separator
* **write(String s, int off, int len)**
* writes a String from position off for length len
* **write(char [ ] array, int off, int len)**
* writes a character array from position off for length len
* **write(int c)**
* writes single character

