Lecture 13

Java Files

- File handling is an important part of any application.
- Java has several methods for creating, reading, updating, and deleting files.
- The **File** class from the **java.io** package, allows us to work with files.
- To use the **File** class, create an object of the class, and specify the filename or directory name.

```
import java.io.File;  // Import the File class
File myObj = new File("filename.txt"); // Specify the filename
```

File Method

Method	Type	Description
canRead()	Boolean	Tests whether the file is readable or not
canWrite()	Boolean	Tests whether the file is writable or not
createNewFile()	Boolean	Creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	Tests whether the file exists

File Method

Method	Type	Description
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	Long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the directory
mkdir()	Boolean	Creates a directory

Create a File

- To create a file in Java, you can use the createNewFile() method. This method returns a boolean value: true if the file was successfully created, and false if the file already exists.

- Note that the method is enclosed in a try...catch block. This is necessary because it throws an IOException if an error occurs (if the file cannot be created for some reason)

Create a File

- To create a file in a specific directory (requires permission), specify the path of the file and use double backslashes to escape the "\" character (for Windows). On Mac and Linux you can just write the path, like: /Users/name/filename.txt

```
File myObj = new File("C:\\Users\\MyName\\filename.txt");
```

Write To a File

- In the following example, we use the **FileWriter** class together with its **write()** method to write some text to the file we created in the example above.
- Note that when you are done writing to the file, you should close it with the **close()** method:

Read Files

- we use the **Scanner** class to read the contents of the text file

- There are many available classes in the Java API that can be used to read and write files in Java: **FileReader, BufferedReader, Files, Scanner, FileInputStream, FileWriter, BufferedWriter, FileOutputStream**, etc. Which one to use depends on the Java version you're working with and whether you need to read bytes or characters, and the size of the file/lines etc.

Delete

- To delete a file in Java, use the **delete()** method.
- You can also delete a folder. However, it must be empty.

FileOutputStream

- Java FileOutputStream is an output stream used for writing data to a **file**.
- If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use **FileWriter** than FileOutputStream.

```
public class FileOutputStream extends OutputStream
```

FileInputStream

- Java FileInputStream class obtains input bytes from a **file**. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use **FileReader** class.

  ```
  public class FileInputStream extends InputStream
  ```

BufferedOutputStream

- Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

- For adding the buffer in an OutputStream, use the BufferedOutputStream class.

```
OutputStream os= new BufferedOutputStream(new
 FileOutputStream("D:\\IO Package\\testout.txt"));
```

BufferedInputStream

- Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

- The important points about BufferedInputStream are:

- When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.

- When a BufferedInputStream is created, an internal buffer array is created.

FileWriter

- Java FileWriter class is used to write character-oriented data to a **file**. It is character-oriented class which is used for file handling in Java
- Unlike FileOutputStream class, you don't need to convert string into byte **array** because it provides method to write string directly.

FileReader

- Java FileReader class is used to read data from the file. It returns data in byte format like **FileInputStream** class.

- It is character-oriented class which is used for **file** handling in **java**.