**How do we access different attributes of a file?**

For eg., at what location the file exists? Do we have permission to read/write on file? The name we are giving to a file, is it the name of a directory? Its absolute/ relative path? Access/modified time?

* We get exception when we try to read a file (whose permission is not given). So we need to take some precautionary measures.
* Java provides **java.io.File. File** is name of class.
* If you are trying to access info of any file, create instance of File class.

**File f = new File(“test.txt”);**

* File class has certain functions like exit(), canread(),….

**Why we are trying to access attributes?**

IO Exception occurs when file doesn’t exist or we don’t have permission on file. So first put a check whether file exists or not. By this you are **minimizing the chances of exception.**

**=>**

|  |  |  |
| --- | --- | --- |
| **Functions** | **Description** | **Return Type** |
| f.getName() | Prints name of file | **String** |
| f.exists() | Tells whether file exists or not | **Boolean** |
| f.canRead() | Checks if the file represented by **f** can be read. | **Boolean** |
| f.canWrite() | Checks if the file represented by **f** can be written to | **Boolean** |
| f.getPath() | Returns the path of the file represented by **f**. | **String** (path of file). |
| f.getAbsolutePath() | Returns the absolute path of the file represented by **f.** | **String** (absolute path of the file). |
| f.getAbsoluteFile() | Returns a **File** object representing the absolute path of the file represented by **f** | **File** (File object representing the absolute path of the file). |
| f.toURL() | Converts the file represented by **f** to a **URL** | **URL** (URL representing the file). |
| f.getParent() | Returns the path of the parent directory of the file represented by **f**. | **String** (path of the parent directory). |
| f.isDirectory() | Checks if the file represented by **f** is a directory. | **boolean** (true if the file is a directory, false otherwise). |
| f.isFile() | Checks if the file represented by **f** is a regular file. | **boolean** (true if the file is a regular file, false otherwise). |
| f.isHidden() | Checks if the file represented by **f** is hidden. | **boolean** (true if the file is hidden, false otherwise). |
| f.lastModified() | Returns the time when the file represented by **f** was last modified, measured in milliseconds since the epoch (January 1, 1970, 00:00:00 GMT). | **long** (time in milliseconds). |
| f.length() | Returns the length of the file represented by **f** in bytes. | **long** (file length in bytes). |
| f.list() | Returns an array of strings representing the names of the files and directories in the directory represented by **f**. | **String[]** (array of file and directory names). |

Note: FileTest.java uses or overrides a deprecated API.

Note: Recompile with -Xlint:deprecation for details. It means?

The message "Note: FileTest.java uses or overrides a deprecated API" indicates that the Java compiler has detected the use or overriding of a deprecated API in the **FileTest.java** file.

1. **Deprecated API**:
   * A deprecated API is a part of a programming interface (such as a class, method, or field) that has been marked as outdated or obsolete and is no longer recommended for use.
   * Deprecated APIs are typically kept in the Java platform for backward compatibility reasons, but developers are encouraged to migrate away from them to newer, more efficient, or more secure alternatives.
2. **-Xlint:deprecation**:
   * This message suggests that you can recompile the code with the **-Xlint:deprecation** option to get more detailed information about the usage of deprecated APIs.
   * The **-Xlint:deprecation** option is a compiler flag that enables additional warnings related to the use of deprecated APIs. When enabled, the compiler will provide more specific information about where and how deprecated APIs are used in the code.
3. **Reasons to Recompile with -Xlint:deprecation**:
   * By recompiling with **-Xlint:deprecation**, you can get a more comprehensive report on where deprecated APIs are used in your codebase.
   * This can help you identify and address deprecated API usage, allowing you to update your code to use modern, supported alternatives.
   * It ensures that your codebase remains up-to-date, maintains compatibility with future Java versions, and follows best practices in software development.

In summary, the message suggests that the code in **FileTest.java** is using or overriding deprecated APIs, and recompiling with the **-Xlint:deprecation** option can provide more detailed information to address these deprecated API usages.