CISC 3115
Java API Classes: File and Path

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Discussed
  - Approaches to handle errors (what-if and exceptions)
  - Concept of Exception
  - The Java throwable class hierarchy
    - system errors, runtime exceptions, checked errors, unchecked errors
  - Methods of declaring, throwing, catching exception, and rethrowing exceptions
  - Exception, call stack, stack frame, and stack trace
  - Some best practice
- Exception and simple text/character File I/O
  - (discussed) File system path (to identify file)
  - Concept of text file (Java API classes and text file)
  - Reliable processing text file (patterns and exceptions)

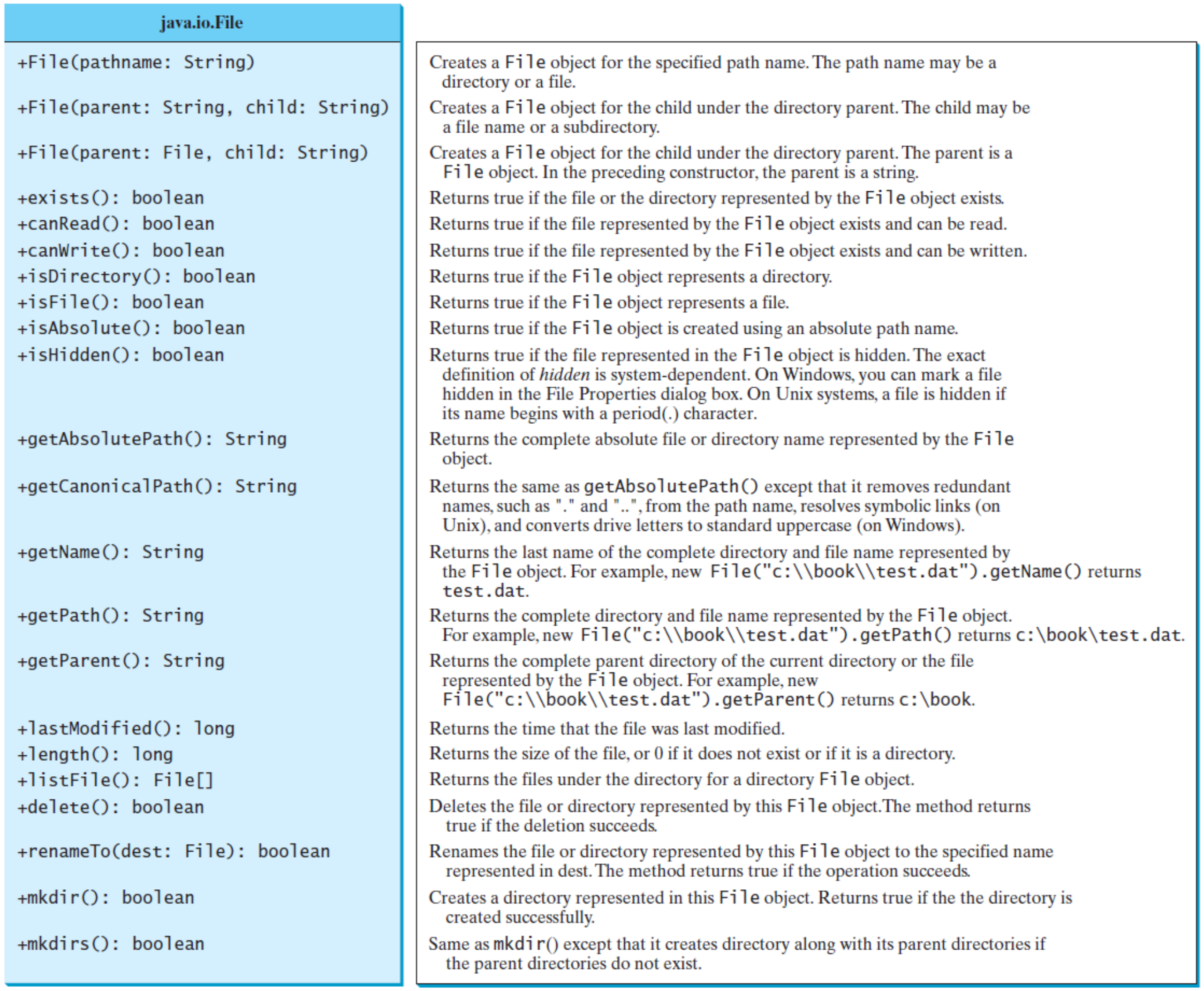
Identifying a file using Java API

- The File class (in the java.io package)
- The Path interface, Paths helper class, and Files helper class (in the java.nio.file package)
  - What is an “interface”? Treat it as a “class” for now.

The File Class

- java.io.File
  - It provides an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion.
  - It is a wrapper class for the file name and its file system path.
  - The filename and its file system path are strings.

The File Class: API



java.io.File	
`+File(pathname: String)`	Creates a `File` object for the specified path name. The path name may be a directory or a file.
`+File(parent: String, child: String)`	Creates a `File` object for the child under the directory parent. The child may be a file name or a subdirectory.
`+File(parent: File, child: String)`	Creates a `File` object for the child under the directory parent. The parent is a `File` object. In the preceding constructor, the parent is a string.
`+exists(): boolean`	Returns true if the file or the directory represented by the `File` object exists.
`+canRead(): boolean`	Returns true if the file represented by the `File` object exists and can be read.
`+canWrite(): boolean`	Returns true if the file represented by the `File` object exists and can be written.
`+isDirectory(): boolean`	Returns true if the `File` object represents a directory.
`+isFile(): boolean`	Returns true if the `File` object represents a file.
`+isAbsolute(): boolean`	Returns true if the `File` object is created using an absolute path name.
`+isHidden(): boolean`	Returns true if the file represented in the `File` object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period(.) character.
`+getAbsolutePath(): String`	Returns the complete absolute file or directory name represented by the `File` object.
`+getCanonicalPath(): String`	Returns the same as `getAbsolutePath()` except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows).
`+getName(): String`	Returns the last name of the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getName()` returns `test.dat`.
`+getPath(): String`	Returns the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getPath()` returns `c:\book\test.dat`.
`+getParent(): String`	Returns the complete parent directory of the current directory or the file represented by the `File` object. For example, new `File("c:\\book\\test.dat").getParent()` returns `c:\book`.
`+lastModified(): long`	Returns the time that the file was last modified.
`+length(): long`	Returns the size of the file, or 0 if it does not exist or if it is a directory.
`+listFile(): File[]`	Returns the files under the directory for a directory `File` object.
`+delete(): boolean`	Deletes the file or directory represented by this `File` object.The method returns true if the deletion succeeds.
`+renameTo(dest: File): boolean`	Renames the file or directory represented by this `File` object to the specified name represented in dest. The method returns true if the operation succeeds.
`+mkdir(): boolean`	Creates a directory represented in this `File` object. Returns true if the the directory is created successfully.
`+mkdirs(): boolean`	Same as `mkdir()` except that it creates directory along with its parent directories if the parent directories do not exist.

Example Problem: Explore File Properties

- Objective
  - Write a program that demonstrates how to create files in a platform-independent way and use the methods in the File class to obtain their properties.
- Observe the example

Example Problem: Explore File Properties

```
public class TestFileClass {

  public static void main(String[] args) {

    java.io.File file = new
java.io.File("image/us.gif");

    System.out.println("Does it exist? " +
file.exists());

    System.out.println("The file has " +
file.length() + " bytes");

    System.out.println("Can it be read? " +
file.canRead());

    System.out.println("Can it be written? " +
file.canWrite());

    System.out.println("Is it a directory? " +
file.isDirectory());

System.out.println("Is it a file? " +
file.isFile());

    System.out.println("Is it absolute? " +
file.isAbsolute());

    System.out.println("Is it hidden? " +
file.isHidden());

    System.out.println("Absolute path is " +

      file.getAbsolutePath());

    System.out.println("Last modified on " +

      new java.util.Date(file.lastModified()));

  }

}
```

File vs. Path

- The Path interface, Paths helper class, and Files helper class
  - Defined in the java.nio.file package (nio stands for “new I/O)
- The Path interface, Paths helper class, and Files helper class (in the java.nio.file package)

Identifying a file using Java API

- The Path interface, Paths helper class, and Files helper class (in the java.nio.file package)
  - What is an "interface"? Treat it as a "class" for now.
- Some shortcomings of the design of the File class (in the java.io package)
  - Poor error handling;
  - Limited meta data support, e.g., permissions, ownership, security attributes (explore on your own);
  - Not performance optimized (explore on your own)

File vs. Path: Error Handling (1)

```
File file = new File("Hw1.txt");
boolean success = file.delete();
```

vs.

```
Path path = Paths.get("Hw1.txt");
Files.delete(path);
```

File vs. Path: Error Handling (2)

```
File file = new File("Hw1.txt");
File[] fileList = file.listFiles();
```

vs.

```
Path path = Paths.get("Hw1.txt");
DirectoryStream<Path> paths =
    Files.newDirectoryStream(path);
```

Using File and Path: Create Instances

- Create File/Path instance

```
java.io.File file = new java.io.File(“Hw1.txt");

java.nio.file.Path path = java.nio.file.Paths.get(“Hw1.txt");


java.io.File file = new File(“alice", “Hw1.txt");

java.nio.Path path = java.nio.file.Paths.get(“alice", “Hw1.txt");
```

Using File and Path: Converting

- Converting between File/Path

```
java.nio.Path pathFromFile = file.toPath();

java.io.File fileFromPath = path.toFile();
```

Using File and Path: Create File and Directory

- Create file

```
boolean success  = file.createNewFile();

java.nio.Path newPath = java.nio.Files.createFile(path);
```

- Create directory

```
boolean success = file.mkdir();

java.nio.Path newPath = java.nio.Files.createDirectory(path);
```

- Create directories along the path

```
boolean result = file.mkdirs();

java.nio.Path newPath = java.nio.Files.createDirectories(path);
```

Using File and Path: Rename or Move or Delete File

- Rename or move file

```
boolean success = file.renameTo(new java.io.File(“Hw2.txt"));

java.nio.Path newPath = java.nio.Files.move(path, Paths.get(“bob/Hw1.txt"));
```

- Delete file

```
boolean success = file.delete();

java.nio.Files.delete(path);
```

Using File and Path: Metadata

- Reading supported metadata (java.io)

```
boolean fileExists = file.exists();
boolean fileIsFile = file.isFile();
boolean fileIsDir = file.isDirectory();
boolean fileReadable = file.canRead();
boolean fileWritable = file.canWrite();
boolean fileExecutable = file.canExecute();
boolean fileHidden = file.isHidden();
```

- Reading supported metadata (java.nio)

```
boolean pathExists = Files.exists(path);
boolean pathIsFile = Files.isRegularFile(path);
boolean pathIsDir = Files.isDirectory(path);
boolean pathReadable = Files.isReadable(path);
boolean pathWritable = Files.isWritable(path);
boolean pathExecutable = Files.isExecutable(path);
boolean pathHidden = Files.isHidden(path);
```

Using File and Path: Path Names

- Get get absolute or canonical paths

```
String absolutePathStr = file.getAbsolutePath();

String canonicalPathStr = file.getCanonicalPath();


absolutePath = path.toAbsolutePath();

Path canonicalPath = path.toRealPath().normalize();
```

Using File and Path: List Directory Content

- List directory content

```
String[] list = file.list();


File[] files = file.listFiles();


DirectoryStream<Path> paths = Files.newDirectoryStream(path);
```

Example Problem: Explore File Properties

- Rewrite the file property exploration example using java.nio

Questions?

- Representing file in Java
- java.io.File, java.nio.Path, java.nio.Paths, and java.nio.Files