# 20251105 Wednesday

- Review exception handling


- New hw

- Test #2


- <mark>Login lab computer now.</mark>

# 20251105

- All time is no time when it is past.



- Scanner

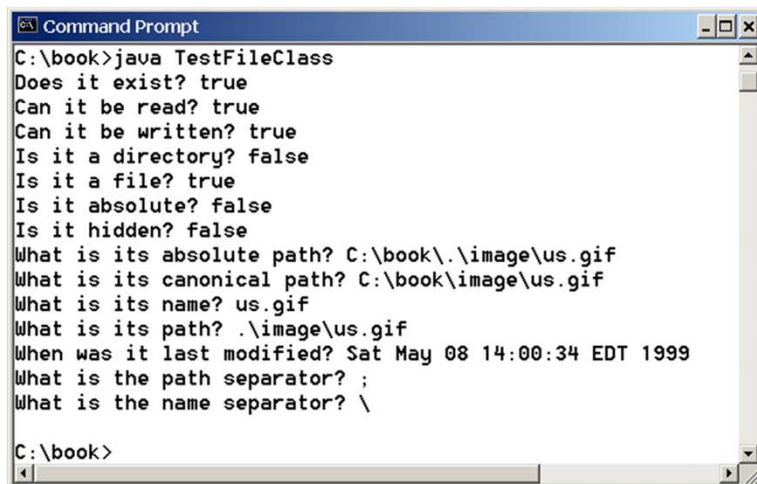- System.out.println()

# The File Class

The <u>File</u> class is intended to provide an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion. The filename is a string. The <u>File</u> class is a wrapper class for the file name and its directory path.

# Obtaining File Properties and Manipulating File

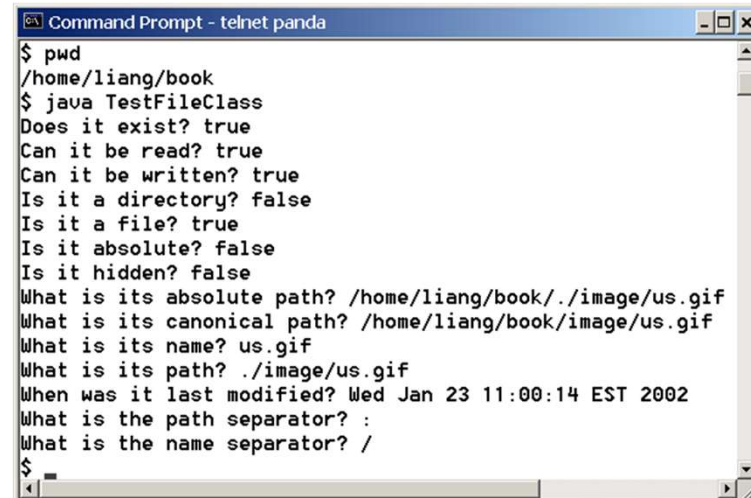| java.io.File | |
|---|---|
| +File(pathname: String) | Creates a File object for the specified path name. The path name may be a directory or a file. |
| +File(parent: String, child: String) | Creates a File object for the child under the directory parent. The child may be a file name or a subdirectory. |
| +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. |

# 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. The following figures show a sample run of the program on Windows and on Unix.

```
Command Prompt                                    _ □ ×
C:\book>java TestFileClass
Does it exist? true
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? false
Is it hidden? false
What is its absolute path? C:\book\.\image\us.gif
What is its canonical path? C:\book\image\us.gif
What is its name? us.gif
What is its path? .\image\us.gif
When was it last modified? Sat May 08 14:00:34 EDT 1999
What is the path separator? ;
What is the name separator? \

C:\book>
```

```
Command Prompt - telnet panda                     _ □ ×
$ pwd
/home/liang/book
$ java TestFileClass
Does it exist? true
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? false
Is it hidden? false
What is its absolute path? /home/liang/book/./image/us.gif
What is its canonical path? /home/liang/book/image/us.gif
What is its name? us.gif
What is its path? ./image/us.gif
When was it last modified? Wed Jan 23 11:00:14 EST 2002
What is the path separator? :
What is the name separator? /
$
```

TestFileClass

# TestFileClass.java

```java
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()));
    }
}
```

```
Does it exist? false
The file has 0 bytes
Can it be read? false
Can it be written? false
Is it a directory? false
Is it a file? false
Is it absolute? false
Is it hidden? false
Absolute path is /home/compiler/image/us.gif
Last modified on Thu Jan 01 00:00:00 GMT 1970
```

# Text I/O

A <u>File</u> object encapsulates the properties of a file or a path but does not contain the methods for reading/writing data from/to a file.

To perform I/O, you need to create objects using appropriate Java I/O classes. The objects contain the methods for reading/writing data from/to a file.

This section introduces how to read/write strings and numeric values from/to a text file using the <mark><u>Scanner</u></mark> and <mark><u>PrintWriter</u></mark> classes.

# Writing Data Using PrintWriter

| java.io.PrintWriter | |
|---|---|
| +PrintWriter(filename: String) | Creates a PrintWriter for the specified file. |
| +print(s: String): void | Writes a string. |
| +print(c: char): void | Writes a character. |
| +print(cArray: char[]): void | Writes an array of character. |
| +print(i: int): void | Writes an int value. |
| +print(l: long): void | Writes a long value. |
| +print(f: float): void | Writes a float value. |
| +print(d: double): void | Writes a double value. |
| +print(b: boolean): void | Writes a boolean value. |
| Also contains the overloaded println methods. | A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix. |
| Also contains the overloaded printf methods. | The printf method was introduced in §4.6, "Formatting Console Output and Strings." |

## WriteData

Pearson

# WriteData.java

```java
public class WriteData {
    public static void main(String[] args) throws java.io.IOException {
        java.io.File file = new java.io.File("scores.txt");
        if (file.exists()) {
            System.out.println("File already exists");
            System.exit(1);
        }

        // Create a file
        java.io.PrintWriter output = new java.io.PrintWriter(file);

        // Write formatted output to the file
        output.print("John T Smith ");
        output.println(90);
        output.print("Eric K Jones ");
        output.println(85);

        // Close the file
        output.close();
    }
}
```

# Try-with-resources

<mark>Programmers often forget to close the file.</mark>
JDK 7 provides the followings new try-with-resources syntax that automatically closes the files.

**try** (declare and create resources) {

  Use the resource to process the file;

}

WriteDataWithAutoClose

# Reading Data Using Scanner

| java.util.Scanner | |
|---|---|
| +Scanner(source: File) | Creates a Scanner object to read data from the specified file. |
| +Scanner(source: String) | Creates a Scanner object to read data from the specified string. |
| +close() | Closes this scanner. |
| +hasNext(): boolean | Returns true if this scanner has another token in its input. |
| +next(): String | Returns next token as a string. |
| +nextByte(): byte | Returns next token as a byte. |
| +nextShort(): short | Returns next token as a short. |
| +nextInt(): int | Returns next token as an int. |
| +nextLong(): long | Returns next token as a long. |
| +nextFloat(): float | Returns next token as a float. |
| +nextDouble(): double | Returns next token as a double. |
| +useDelimiter(pattern: String): Scanner | Sets this scanner's delimiting pattern. |

ReadData

P Pearson

# ReadData.java

```java
import java.util.Scanner;
public class ReadData {
    public static void main(String[] args) throws Exception {
        // Create a File instance
        java.io.File file = new java.io.File("scores.txt");
        // Create a Scanner for the file
        Scanner input = new Scanner(file);

        // Read data from a file
        while (input.hasNext()) {
            String firstName = input.next();
            String mi = input.next();
            String lastName = input.next();
            int score = input.nextInt();
            System.out.println( firstName + " " + mi + " " + lastName + " " + score );
        }

        // Close the file
        input.close();
    }
}
```

# Review

- "File" class to manipulate files

- "PrintWriter" class to print/output to a file

- "Scanner" class to input from a file

# Practice 1/4

- on lab computer, use the command line interface "cmd"
- 1 `cd Documents`
- 2 `mkdir CSC1850_20251105`
- 3 `cd CSC1850_20251105`
- 4 `notepad TestFileClass.java`
- 5 `javac TestFileClass.java`
- 6 `java TestFileClass`

# Practice 2/4

- Similar as previous slide, practice with "WriteData.java"

- Please be creative to add more data to write to the file.


- Please open and check the file "scores.txt"

# Practice 3/4

- practice with "ReadData.java"

- Is the output what you expect?

# Practice 4/4

- Now, run the "WriteData" again. What happen?

- Please delete the "scores.txt", and run the "ReadData" again, What happen?

- HW please.

Pearson