CS112

File Class
Chapter 12

Lecture 11

الفصل الدراسي الثاني 1443- 2022 Spring

College of Computer Science and Engineering

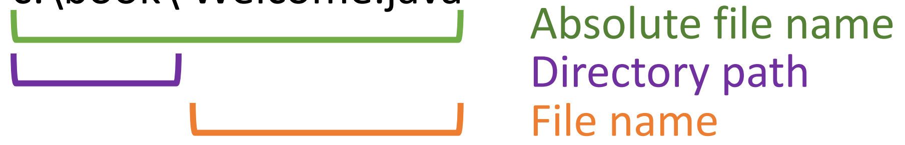
جامعة طيبة

Introduction

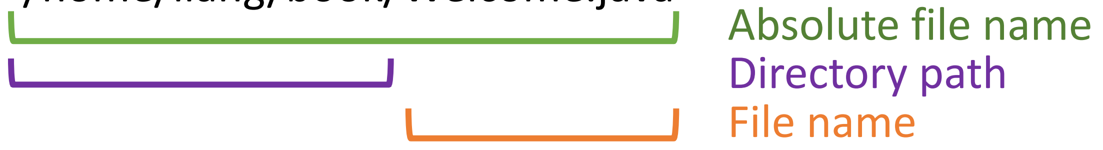
- The <u>File</u> class is intended to provide an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion.
- The filename is a string.
- The <u>File</u> class is a wrapper class for the file name and its directory path.

Absolute File Name

- Every file is placed in a directory in the file system.
- An absolute file name (or full name) contains a file name with its complete path and drive letter.
  - For example (Windows):
    - c:\book\ Welcome.java



  - For example (UNIX):
    - /home/liang/book/Welcome.java



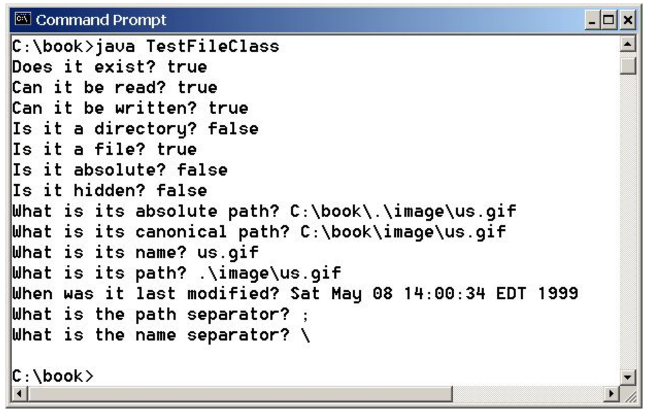
The directory separator for Windows is a backslash (\). The backslash is a special character in Java and should be written as \\ in a string literal (see Table 4.5 in textbook)

Relative File Name

- A relative file name is in relation to the current working directory.
- The complete directory path for a relative file name is omitted.
  - For example, Welcome.java is a relative file name. If the current working directory is c:\book, the absolute file name would be c:\book\Welcome.java

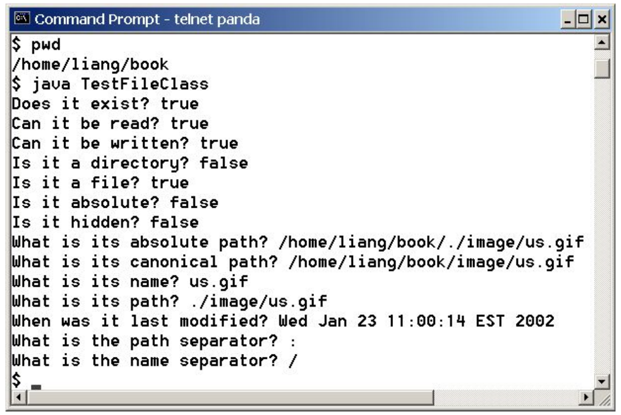
Obtaining File Properties and Manipulating File

- Exercise:
  - 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.
    - See TestFileClass.java

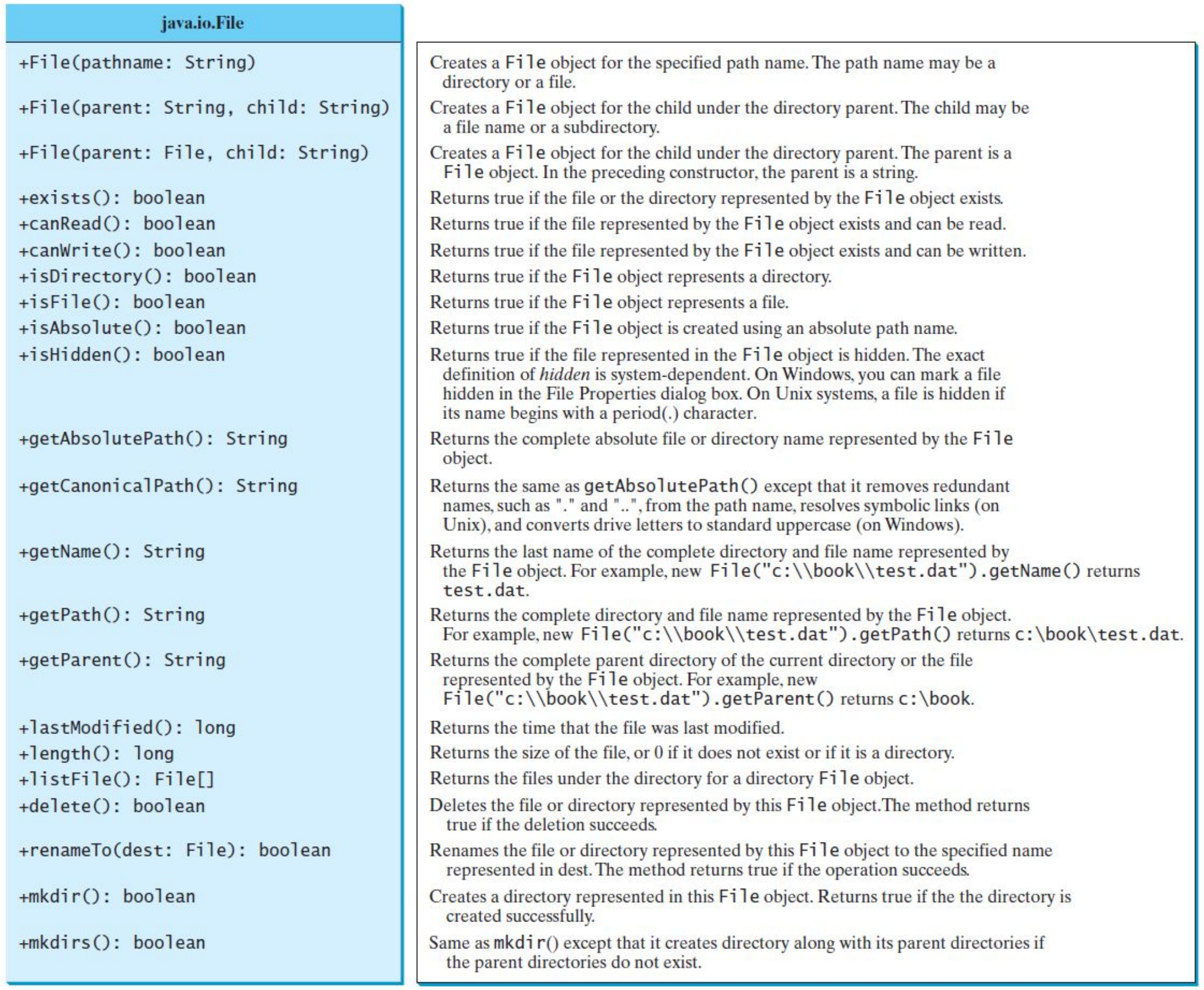

Command Prompt

```
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? /
$ _
```



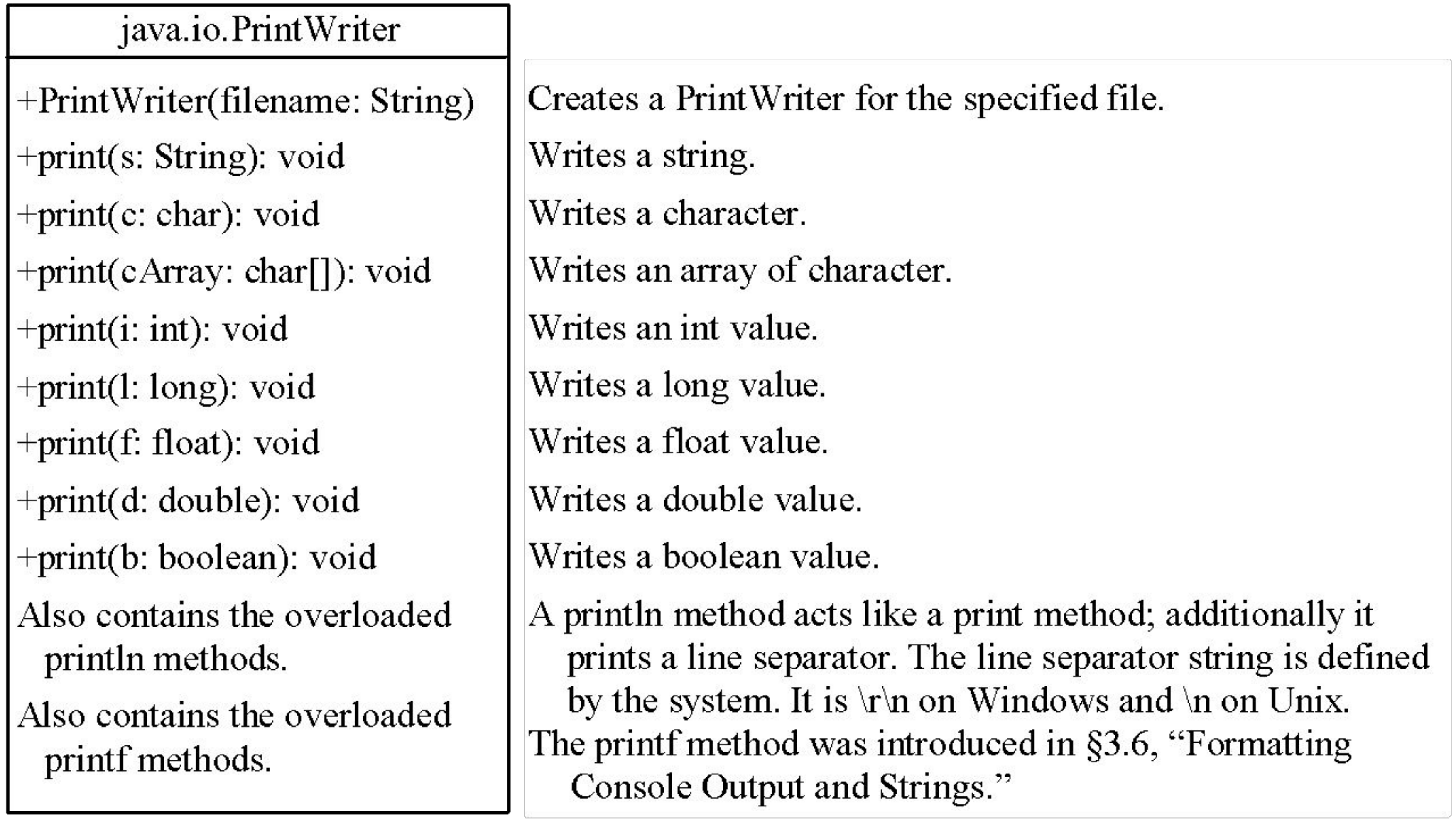
java.io.File	
+File(pathname: String)	Creates a `File` object for the specified path name. The path name may be a directory or a file.
+File(parent: String, child: String)	Creates a `File` object for the child under the directory parent. The child may be a file name or a subdirectory.
+File(parent: File,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

File I/O

- A File object encapsulates the properties of a file or a path, but does not contain the methods for reading/writing data from/to a file.
- In order 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.
- Now, let’s see how to read/write strings and numeric values from/to a text file using the Scanner and PrintWriter classes.

Writing Data Using PrintWriter

- See WriteData.java



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 §3.6, "Formatting Console Output and Strings."

Try-with-resources

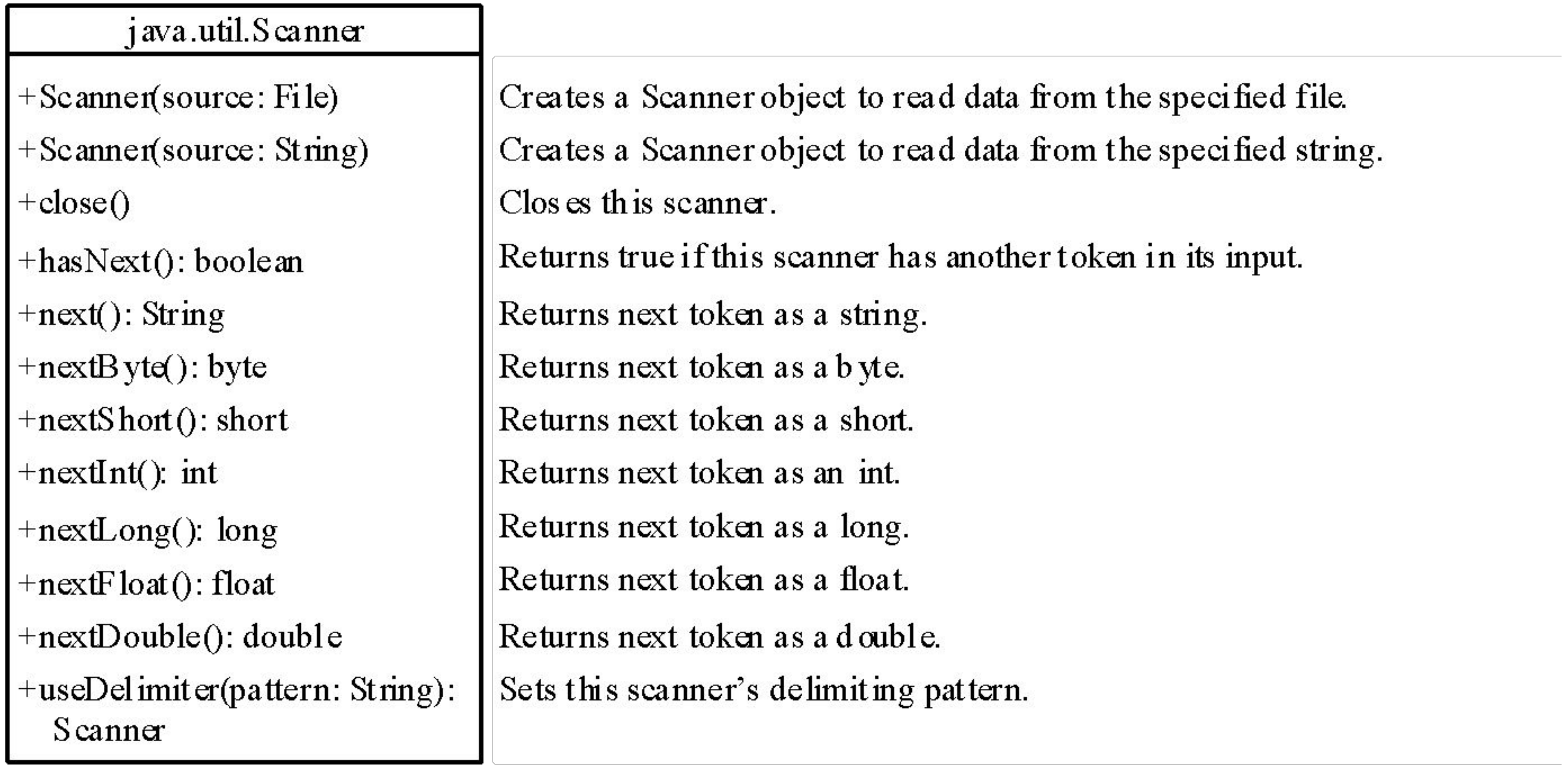
- Programmers often forget to close the file.
- JDK 7 provides the followings new try-with-resources syntax that automatically closes the files.

```
try (declare and create resources) {
  Use the resource to process the file;
}
```

- See WriteDataWithAutoClose.java

Reading Data Using Scanner

- See ReadData. java



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.

Case Study: Replacing Text

- Write a class named ReplaceText that replaces a string in a text file with a new string. The filename and strings are passed as command-line arguments as follows:

```
java ReplaceText sourceFile targetFile oldString newString
```

For example, invoking

```
java ReplaceText FormatString.java t.txt StringBuilder StringBuffer
```

replaces all the occurrences of StringBuilder by StringBuffer in FormatString.java and saves the new file in t.txt.

- See ReplaceText.java and TestReplaceText.java

Recommended Readings

- Reading data from the Web:
  - Section 12.12 and 12.13 in textbook