CISC 3115 TY3

C17a: Exception and Text File I/O

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Discussed
  - Error and error handling
    - Two approaches
  - Exception
  - The throwable class hierarchy
    - System errors and semantics
    - Runtime exceptions and semantics
    - Checked errors and semantics
  - Declaring, throwing, and catching exception
  - Exception, call stack, stack trace, the finally clause, and rethrowing exceptions
  - Custom exceptions
- Exception and simple text/character File I/O

Path and File

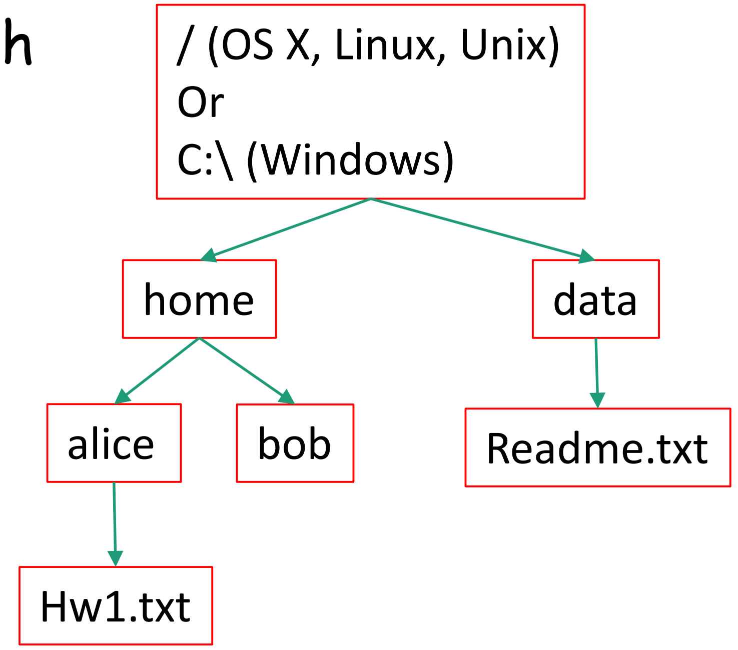
- Concept of path in OS
- The Path interface and Paths helper class
- The File and Files classes

File System Trees

- A file system stores and organizes files on some form of media allowing easy retrieval
- Most file systems in use store the files in a tree (or hierarchical) structure.
  - Root node at the top
  - Children are files or directories (or folders in Microsoft Windows)
  - Each directory/folder can contain files and subdirectories

Path

- Identify a file by its *path* through the file system tree, beginning from the root node
  - Example: identify Hw1.txt
  - OS X
    - /home/alice/Hw1.txt
  - Windows
    - C:\home\alice\Hw1.txt
  - Delimiter
    - Windows: "\"
    - Unix-like: "/"

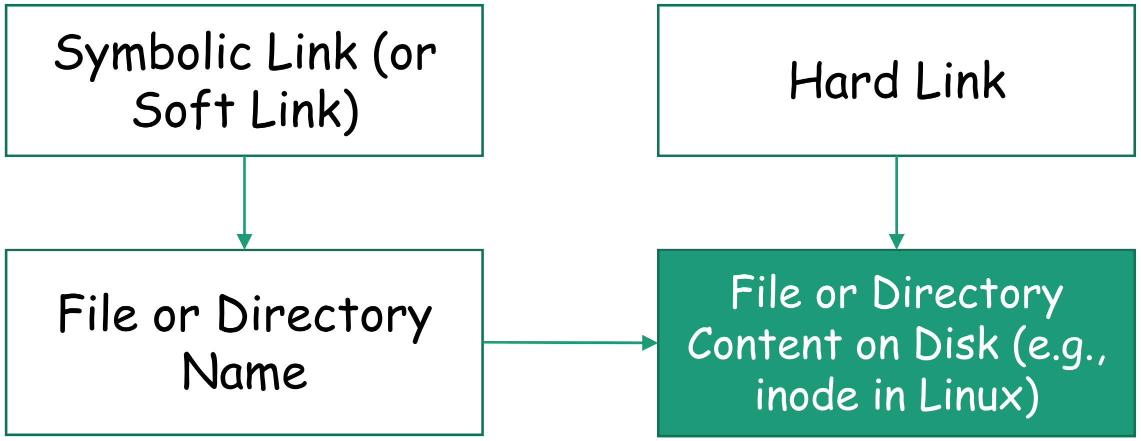


Relative and Absolute Path

- Absolute path
  - Contains the root element and the complete directory list required to locate the file
    - Example: /home/alice/Hw1.txt or C:\home\alice\Hw1.txt
- Relative path
  - Needs to be combined with another path in order to access a file.
  - Example
    - alice/Hw1.txt or alice\Hw1.txt, without knowing where alice is, a program cannot locate the file
  - "." is the path representing the current working directory
  - ".." is the path representing the parent of the current working directory

Symbolic Link and Hard Link

- A file-system object (source) that points to another file system object (target).
  - Symbolic link (soft link): an "alias" to a file or directory name
  - Hard link: another name of a file or directory


Symbolic Link (or Soft Link)
Hard Link
File or Directory Name
File or Directory Content on Disk (e.g., inode in Linux)

Transparency to Users

- Links are transparent to users
  - The links appear as normal files or directories, and can be acted upon by the user or application in exactly the same manner.
- Create symbolic links from the Command Line
  - Unix-like: ln
  - Windows: mklink

Unix-like OS: Example

- Unix-like (e.g., Linux, OS X): “#” leads a comment. do the following on the terminal,

```
echo “hello, world!” > hello.txt      # create a file, the content is “hello, world!”
ln -s hello.txt hello_symlink.txt     # create a soft link to hello.txt
ls -l hello_symlink.txt               # list the file, what do we observe?
cat hello_symlink.txt                 # show the content using the symbolic link, what do we observe?
ln hello.txt hello_hardlink.txt       # create a hard link
ln -l hello_hardlink.txt              # observation?
cat hello_hardlink.txt                # observation?
mv hello.txt hello2.txt               # rename hello.txt
ls -l hello_symlink.txt               # observation?
ln -l hello_hardlink.txt              # observation?
cat hello_symlink.txt                 # observation?
cat hello_hardlink.txt                # observation
```

Window: Example

- On Windows, it requires elevated privilege to create file symbolic link. Do not type the explanation in “()”.

echo “hello, world!” > hello.txt	(create a file, the content is “hello, world!”)
mklink hello_symlink.txt hello.txt	(create a soft link to hello.txt)
dir hello_symlink.txt	(list the file, what do we observe?)
more hello_symlink.txt	(show the content using the symbolic link, what do we observe?)
mklink /h hello_hardlink.txt hello.txt	(create a hard link to hello.txt)
dir hello_hardlink.txt	(observation?)
more hello_hardlink.txt	(observation?)
move hello.txt hello2.txt	(rename hello.txt)
dir hello_symlink.txt	(observation?)
dir hello_hardlink.txt	(observation?)
more hello_symlink.txt	(observation?)
more hello_hardlink.txt	(observation?)

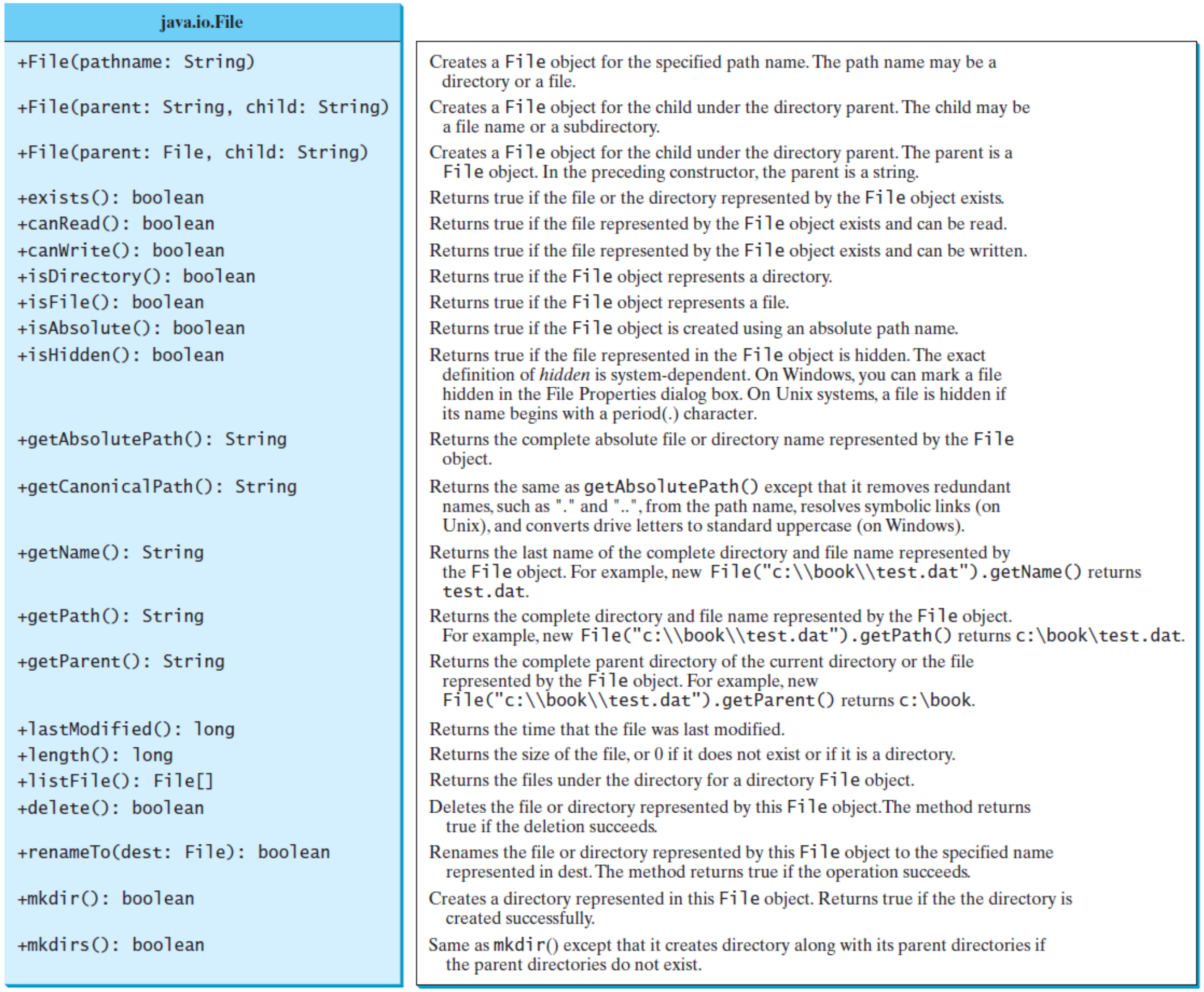
Questions?

- Concept of file system trees
- Concept of paths
  - Traversal of file system trees
  - Absolute path
  - Relative path
- Symbolic link and hard link

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 directory path.
    - The filename is a string.

The File Class: API

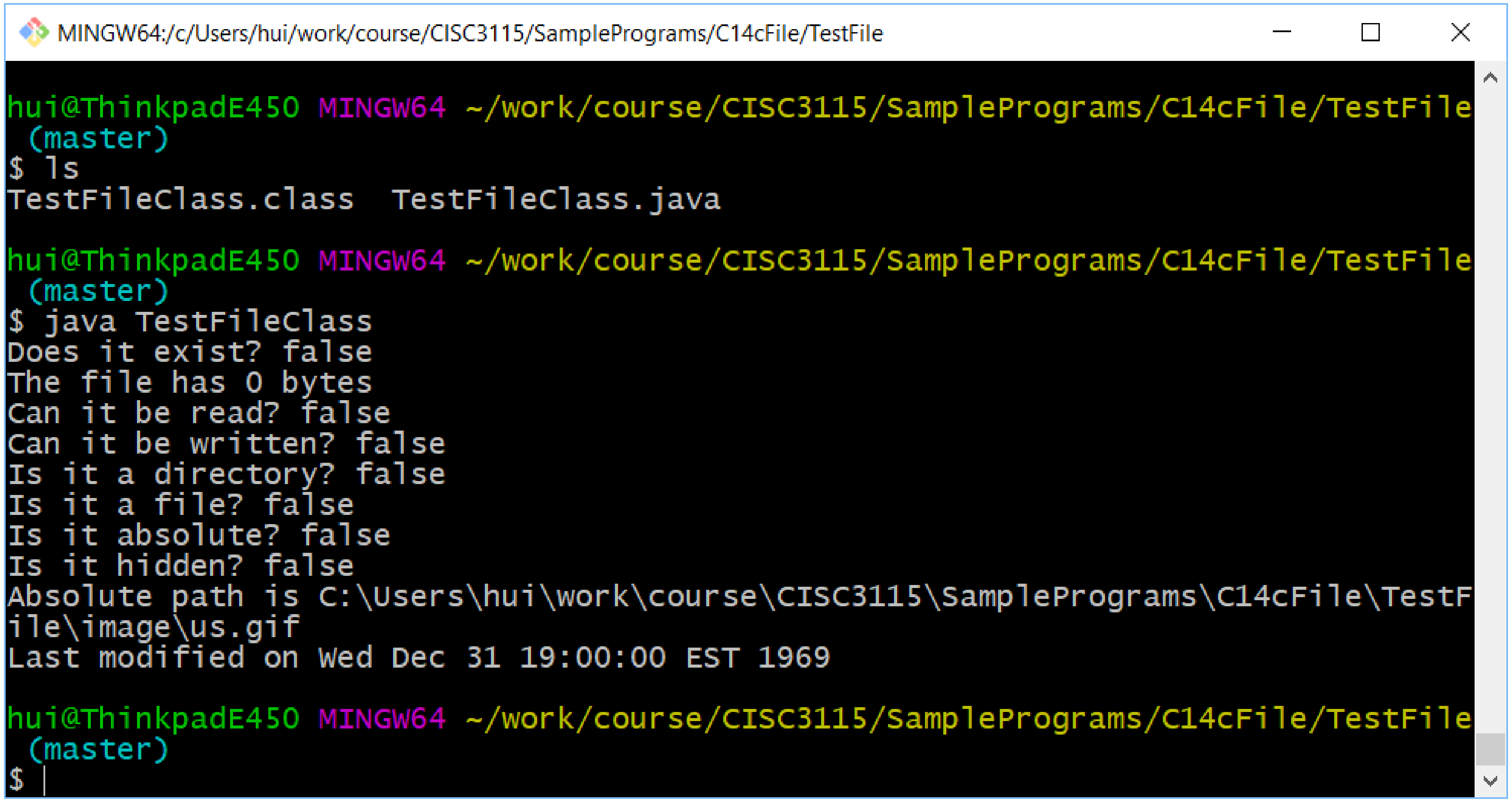


java.io.File	
`+File(pathname: String)`	Creates a `File` object for the specified path name. The path name may be a directory or a file.
`+File(parent: String, child: String)`	Creates a `File` object for the child under the directory parent. The child may be a file name or a subdirectory.
`+File(parent: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


MINGW64:/c/Users/hui/work/course/CISC3115/SamplePrograms/C14cFile/TestFile

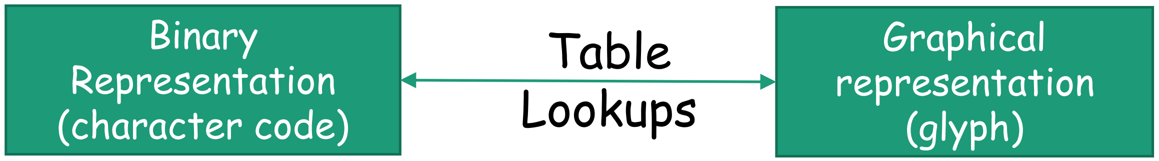
```
hui@ThinkpadE450 MINGW64 ~/work/course/CISC3115/SamplePrograms/C14cFile/TestFile
 (master)
$ ls
TestFileClass.class  TestFileClass.java

hui@ThinkpadE450 MINGW64 ~/work/course/CISC3115/SamplePrograms/C14cFile/TestFile
 (master)
$ java TestFileClass
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 C:\Users\hui\work\course\CISC3115\SamplePrograms\C14cFile\TestF
ile\image\us.gif
Last modified on Wed Dec 31 19:00:00 EST 1969

hui@ThinkpadE450 MINGW64 ~/work/course/CISC3115/SamplePrograms/C14cFile/TestFile
 (master)
$ 
```

Text File

- Also called character file.
- Each stores characters

Characters

- Basic units to form written text
  - Each language has a set of characters
  - Generally, a character is a code (a binary number)
  - A character can have many different glyphs (graphical representation)
    - The 1st letter in the English Alphabet
      - Character "a": a, **a**, **a**, ***a***, …


Binary Representation (character code)
Table Lookups
Graphical representation (glyph)

Unicode

- A single coding scheme for written texts of the world's languages and symbols
- Each character has a code point
    - Originally 16-bit integer (0x0000 – 0xffff), extended to the range of (0x0 – 0x10ffff), *e.g.*, U+0000, U+0001, …, U+2F003, …, U+FF003, …, U+10FFFF
- All the codes form the Unicode code space
    - Divided into planes, each plane is divided into blocks
        - Basic Multilingual Plane (BMP), the 1st plane, where a language occupies one or mote blocks
- Encoding schemes
    - Express a code point in bytes: in UTF-8, use 1 to 4 bytes (grouped into code units) to represent a code point (space saving, backward comparability with ASCII)
    - Code units

Encoding Scheme: Code Point and Code Units: Examples

- All code units are in hexadecimal.

Unicode code point	U+0041	U+00DF	U+6771	U+10400
Representative glyph	A	β	東	∂
UTF-32 code units	00000041	000000DF	00006771	00010400
UTF-16 code units	0041	00DF	6771	D801 DC00
UTF-8 code units	41	C3 9F	E6 9D B1	F0 90 90 80

Characters in the Java Platform

- Original design in Java
  - A character is a 16-bit Unicode
    - A Unicode 1.0 code point is a 16-bit integer
    - Java predates Unicode 2.0 where a code point was extended to the range (0x0 – 0x10ffff).
    - Example: U+0012: '\u0012'
- Evolved design: a character in Java represents a UTF-16 code unit
  - The value of a character whose code point is no above U+FFFF is its code point, a 2-byte integer
  - The value of a character whose code point is above U+FFFF are 2 code units or 2 2-byte integers ((high surrogate: U+D800 ~ U+DBFF and low surrogate: U+DC00 to U+DFFF)
- In Low-level API: Use code point, a value of the int type (e.g., static methods in the Character class)

Text I/O

- The File objects contain the methods for reading/writing data from/to a file.
- Objective: To read/write strings and numeric values from/to a text file using the Scanner and PrintWriter classes.

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."

PrintWriter: close()

- Any system resources associated with a PrintWriter object must be released
- Use PrintWriter::close()

Write Text to File: Example: First Try

- See WriteText.java
- Is there any problem?

```
PrintWriter output = new 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();
```

Write Text to File: Example: First Try: Resources Released?

- See WriteText
- Is there any problem?

```
PrintWriter output = new 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();
```

Exception may occur, resulting in close() method not called.

Write Text to File: Example: Second Try: Close in Finally

- See WriteTextCloseWithFinally

```
PrintWriter output = null;
try {
    output = new PrintWriter(file);
    output.print("John T Smith ");
    output.println(90);
    output.print("Eric K Jones ");
    output.println(85);
    System.out.println("Wrote to " + file.getAbsolutePath());
    System.out.println("Or wrote to relative path " + file.getPath());
  } finally {
    if (output != null) { output.close();  }
}
```

Try-With-Resource

- 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;
}
```

Write Text to File: Example: Second Try: Autoclose

- See WriteTextAutoclose

```
try (PrintWriter output = new PrintWriter(file)) {
    // Write formatted output to the file
    output.print("John T Smith ");
    output.println(90);
    output.print("Eric K Jones ");
    output.println(85);
    System.out.println("Wrote to " + file.getAbsolutePath());
    System.out.println("Or wrote to relative path " + file.getPath());
  }
```

Questions?

- Concept of character and text file
- Concept of file system path and file
- Writing text using File and PrintWriter
    - How to handle exception?
    - What are the approaches to release system resources used by PrintWriter?

Reading Text 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.

Example Problem: Replacing Text

- Problem:
  - 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.

Example Program: Replacing Text

- See ReplaceText

```
try ( // try-with-resource to autoclose resources
    Scanner input = new Scanner(sourceFile);
    PrintWriter output = new PrintWriter(targetFile);
  ) {
    while (input.hasNext()) {
      String s1 = input.nextLine();
      String s2 = s1.replaceAll(args[2], args[3]);
      output.println(s2);
    }
  }
```

Questions?

- Use Scanner to read text file

Exercise C17a-1

- In the ReplaceText example program, we use a try-with-resource to release system resources associated with the Scanner and PrintWriter objects.
  - Create a directory in your weekly programming repository , and the directory's name match the exercise number.
  - Revise the class to release resources in the finally block
  - In ReplaceText, we declare the main(String[] args) method to throw Exception. Revise the program so that exceptions are handled in the main method by using the catch clause.
    - However, you catch as specific type exception as you can.
  - Use git to make a submission

Exercise C17a-2

- This is question 12.11 in chapter 12 of the textbook. Write a program that removes all the occurrences of a specified string from a text file. For example, invoking

```
Java ExerciseC17a2 john filename.txt
```

removes the string john from the filename.txt file.

  - Create a directory in your weekly programming repository , and the directory's name match the exercise number.
  - Use the ReplaceText example program as a start
  - In ReplaceText, we declare the main(String[] args) method to throw Exception. Revise the program so that exceptions are handled in the main method by using the catch clause.
    - However, you catch as specific type exception as you can.
  - Use git to make a submission