

CISC 3115 TY3

# 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

# Learning Objectives

- Identifying a file (to write to or to read from)
  - Concept of file system path
  - Path and File
- Understanding characters and text file
- Reading from and writing to text files

# Identifying a file

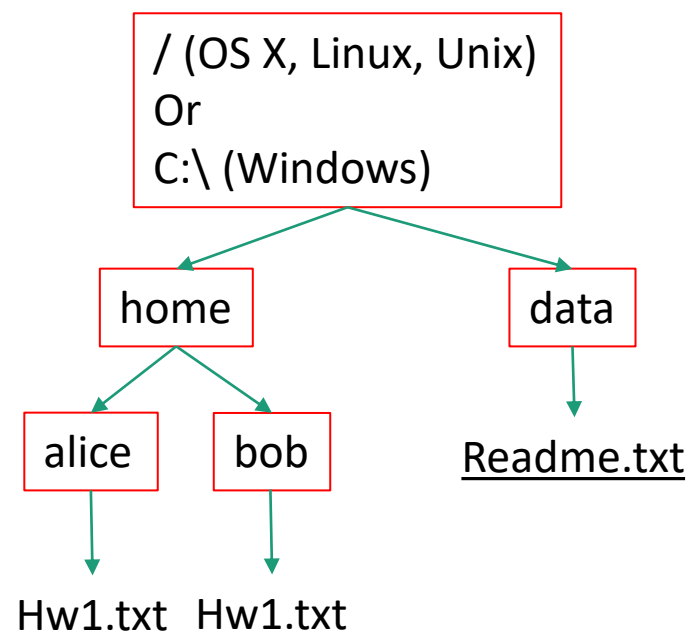
- Concept of path in OS

# 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
  - A path is a “path” of the tree traversal
  - Example: identify Hw1.txt
  - OS X
    - /home/alice/Hw1.txt
  - Windows
    - C:\home\alice\Hw1.txt
  - Delimiter
    - Windows: “\” and “/”
    - Unix-like: “/”
  - Current directory (.) and parent directory (..)



# Relative and Absolute Path

- Absolute path
  - Tree traversal must begin at the root directory
  - 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.
  - The another path is the “reference” (or the beginning directory of the tree traversal), and the reference path isn’t recorded in the path.
  - 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

# Questions?

- Concept of file system trees
- Concept of paths
  - Traversal of file system trees
  - Absolute path
  - Relative path



# 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.
- The [File](#) class (in the java.io package)

# 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 and its directory path are a string.

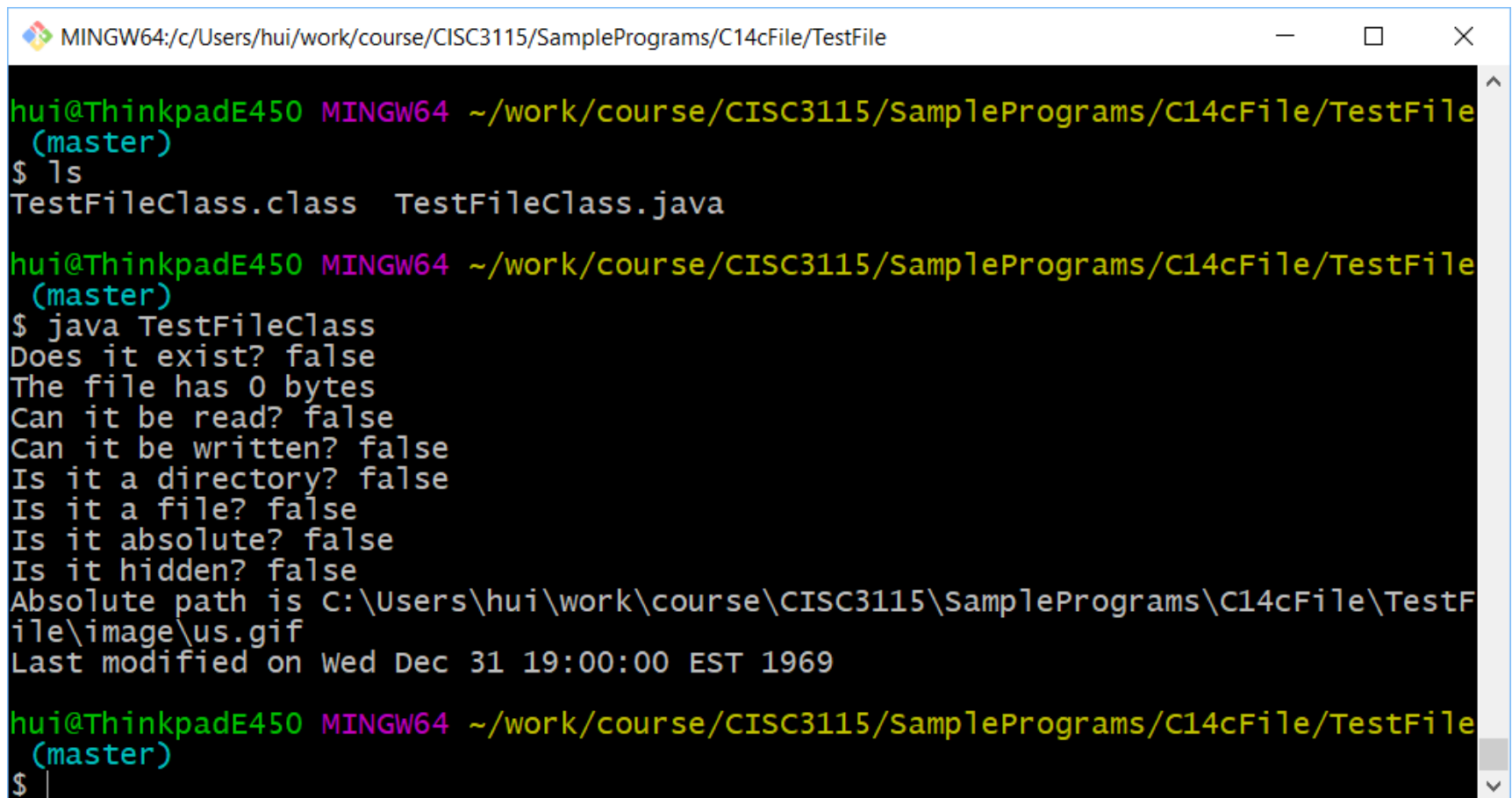
# The File Class: API

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 <i>hidden</i> 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 <code>getAbsolutePath()</code> 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 <code>mkdir()</code> 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



```
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\TestFile\image\us.gif
Last modified on Wed Dec 31 19:00:00 EST 1969

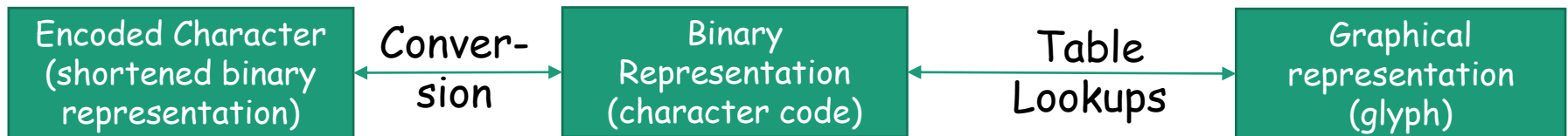
hui@ThinkpadE450 MINGW64 ~/work/course/CISC3115/SamplePrograms/C14cFile/TestFile
(master)
$
```

# Characters and Text File

- Also called character file.
- Each stores characters
- But what are characters

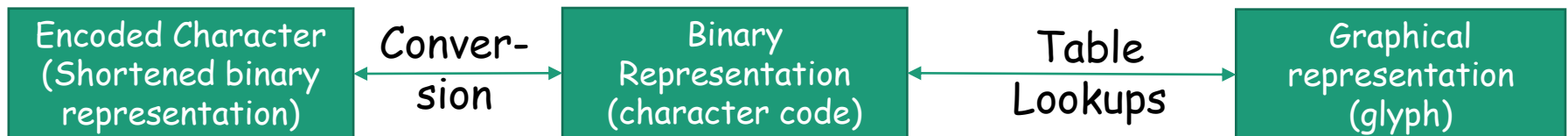
# Characters

- Basic units to form written text
  - Each language has a set of characters
  - Generally, a character is a code (a binary number) in the OS
  - A character can have many different glyphs (graphical representation), provided by a font
    - The 1<sup>st</sup> letter in the English Alphabet
      - Character “a”: a, **a**, **ɑ**, **ɑ**, ...



# 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 1<sup>st</sup> plane, where a language occupies one or more 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

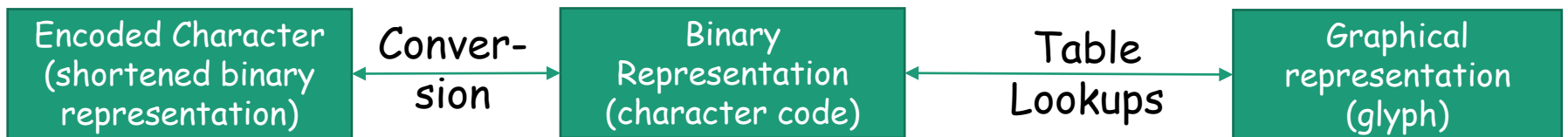




# Encoding Scheme: Code Point and Code Units: Examples

- All code units are in hexadecimal.

Unicode code point	U+0041	U+00DF	U+6771	U+10400
Representative glyph	A	β	東	ð
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



# Let's do some exercises with Unicode codepoints

```
char[][] texts = {  
    Character.toChars(0x00000041), // A  
    Character.toChars(0x000000df), // ß ; on Windows, chcp 850  
    Character.toChars(0x00006771), // 東 ; on windows, chcp 936  
    Character.toChars(0x00000414), // Д ; on windows, chcp 855  
};  
for (int i=0; i<texts.length; i++) {  
    System.out.println(new String(texts[i]));  
}
```

# 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 File

- Also called character file
- Each stores characters
  - Stores encoded binary representations of “characters”
- If we know the encoding scheme, we can *correctly* render the characters in their glyphs
  - What if we don’t know?
- The rest is to introduce Java Text File I/O

# Text File I/O in Java

- 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.
- A few other Java API classes can do text file I/O as well, but leave them for your own exploration

# 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 should be released
- Use the `PrintWriter::close()` method

# Write Text to File: First Try

- Observe 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);  
output.println(63/0);  
// Close the file  
output.close();
```



# Write Text to File: First Try: Resources Always Released?

- Observe 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);  
output.println(63/0);  
// Close the file  
output.close();
```

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

# Write Text to File: Second Try: close() in the finally Block

- Observe WriteText.java
- Is there any problem?

```
PrintWriter output = null;
```

```
try {
```

```
    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); output.println(63/0);
```

```
} finally {
```

```
    // Close the file
```

```
    output.close();
```

```
}
```

# Autoclose using try-with-resources

- 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: Third Try: try-with-resources

```
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);  
  
    output.println(63/0);  
  
}
```

# 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 and Program: 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: the Gist of Replacing Text

```
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

# Exercises 1

- In the ReplaceText example program, we use a try-with-resource to release system resources associated with the Scanner and PrintWriter objects.
  - Create directories in your journal,
  - Revise the class to release resources in the finally block
  - In the ReplaceText example given in the slides, we declare the main(String[] args) method to throw Exception. In this exercise, you must handle exceptions in the main method by using the catch clause.
    - However, you catch as specific types of exceptions as you can.
- Submit the work as a journal entry

# Exercise 2

(This is based on 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 RemoveText john filename.txt
```

removes the string john from the filename.txt file. The rest is similar to exercise 1.

- Create directories in your journal
- Use the RemoveText example program as a start
- In previous exercise, we declare the main(String[] args) method to throw Exception. In this program, you must handle exceptions in the main method by using the catch clause.

However, you catch as specific types of exceptions as you can.

- Submit the work as a journal entry