CISC 3115

Exception and Text File I/O

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

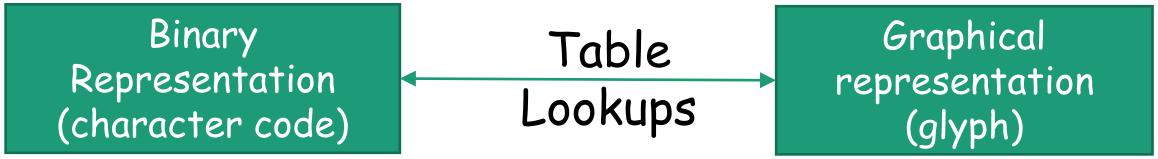
- Discussed
  - Approaches to handle errors (what-if and exceptions)
  - Concept of Exception
  - The Java throwable class hierarchy
    - system errors, runtime exceptions, checked errors, unchecked errors
  - Methods of declaring, throwing, catching exception, and rethrowing exceptions
  - Exception, 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)

Text File

- There is a need to represent text data, i.e., human understandable
- Text file are also called character file
- Store text data
  - written text or binary representations of characters
    - Characters?
    - Binary representations?

Characters

- Basic units to form written text
  - Each language has a set of characters
    - The 1st letter in the English Alphabet is a character
  - On computers, represent characters in bit patterns using character encoding scheme
    - A character is a code (a binary number, binary representation)
    - A character can have many different glyphs (graphical representation)
      - 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 (21 bits)
  - 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

Unicode Code Point Examples

- A code point is 21 bits.
- All codes in these examples are hexadecimal.

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

Unicode Encoding

- Encoding schemes - actual text is processed as binary data via one of several Unicode encodings
  - e.g., UTF-8, UTF-16, UTF-32
  - 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)
    - Character → Unicode code point → Unicode encoding code unit

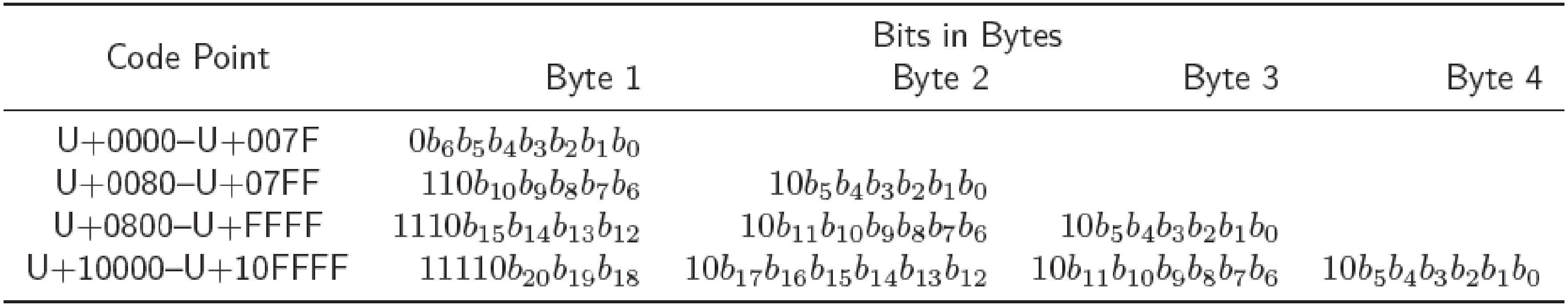
Encoding Scheme: Code Point and Code Units: Examples

- Character → Unicode code point → Unicode encoding code unit
  - For coding scheme like UTF-16, there are variants
    - write the most significant byte first vs. write the most significant byte last

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

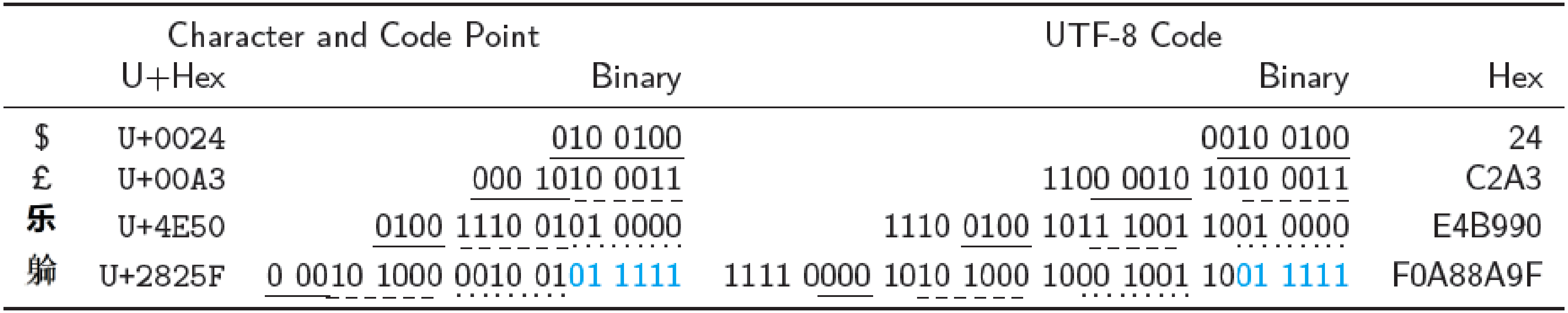
UTF-8

- A variable-length character encoding standard that use 1 to 4 bytes to represent a Unicode character.
- The following table defines the conversion between Unicode code point and variable UTF-8 character bytes



Code Point	Bits in Bytes			
	Byte 1	Byte 2	Byte 3	Byte 4
U+0000–U+007F	$0b_6b_5b_4b_3b_2b_1b_0$			
U+0080–U+07FF	$110b_{10}b_9b_8b_7b_6$	$10b_5b_4b_3b_2b_1b_0$		
U+0800–U+FFFF	$1110b_{15}b_{14}b_{13}b_{12}$	$10b_{11}b_{10}b_9b_8b_7b_6$	$10b_5b_4b_3b_2b_1b_0$	
U+10000–U+10FFFF	$11110b_{20}b_{19}b_{18}$	$10b_{17}b_{16}b_{15}b_{14}b_{13}b_{12}$	$10b_{11}b_{10}b_9b_8b_7b_6$	$10b_5b_4b_3b_2b_1b_0$

UTF-8: Examples



	Character and Code Point		UTF-8 Code	
	U+Hex	Binary	Binary	Hex
$	U+0024	010 0100	0010 0100	24
£	U+00A3	000 1010 0011	1100 0010 1010 0011	C2A3
乐	U+4E50	0100 1110 0101 0000	1110 0100 1011 1001 1001 0000	E4B990
𨉟	U+2825F	0 0010 1000 0010 0101 1111	1111 0000 1010 1000 1000 1001 1001 1111	F0A88A9F

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 Unicode codepoint is now 21 bits.
  - Java uses a UTF-16 code unit to represent a character
  - 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)

Encoding Schemes in Java

- Java supports a few standard encoding schemes for subsets of Unicode characters

Charset	Description
US-ASCII	Seven-bit ASCII, a.k.a. ISO646-US, a.k.a. the Basic Latin block of the Unicode character set
ISO-8859-1	ISO Latin Alphabet No. 1, a.k.a. ISO-LATIN-1
UTF-8	Eight-bit UCS Transformation Format
UTF-16BE	Sixteen-bit UCS Transformation Format, big-endian byte order
UTF-16LE	Sixteen-bit UCS Transformation Format, little-endian byte order
UTF-16	Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark (BOM)

Charset Classes

- Use string (e.g., `"UTF-8"`) or Charset instances to represent an encoding scheme (subsets of Unicode characters)
- java.nio.Charset
  - Defines methods for creating decoders and encoders. Loosely speaking:
    - Encode: Unicode code point → code unit in the encoding scheme
    - Decode: code unit → Unicode code point
  - For retrieving the various names associated with a charset.
  - Instances of this class are immutable.
- java.nio.StandardCharsets
  - Define constant for the standard Charsets.

JVM Define Settings

- Be explicit about the encoding settings. Relying on JVM’s default encoding settings is not recommended.

- Windows

```
C:\> java -XshowSettings 2>&1 |
find "file.encoding"

    file.encoding = Cp1252

C:\> jshell

jshell>
System.getProperty("file.encoding")

$1 ==> "Cp1252"
```

- Linux

```
$ java -XshowSettings 2>&1 | grep
"file.encoding"

    file.encoding = UTF-8

$ jshell

jshell>
System.getProperty("file.encoding")

$1 ==> "UTF-8"
```

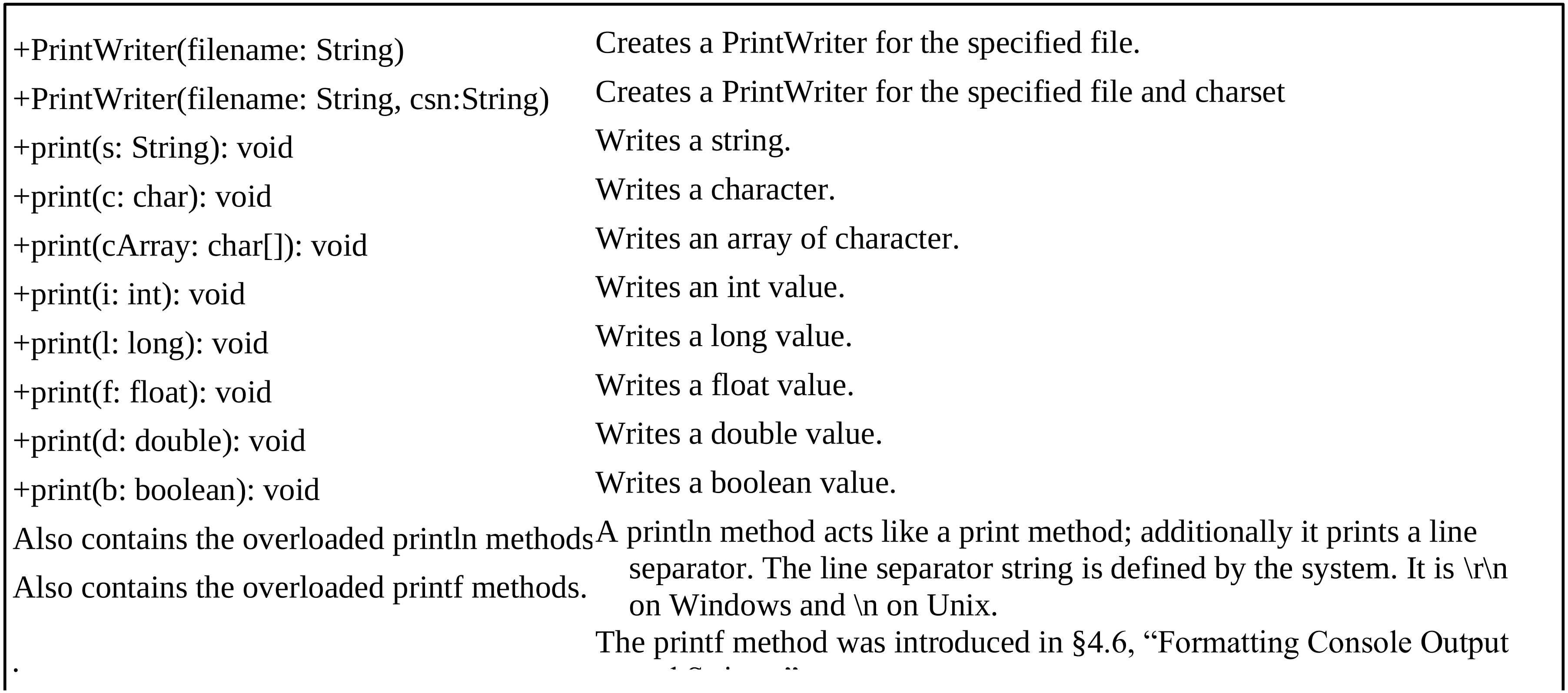
Questions?

- Text data, text file?
  - Characters and strings
- Unicode
  - Codepoint
  - Unicode encoding scheme
    - Code units
- Java?

Text I/O

- Text files or character files contains text data
- Objective
  - To mast the patterns to read/write strings and numeric values from/to a text file using the Scanner and PrintWriter classes.
    - These classes provide a few convenient methods.
  - To process text files in a reliable fashion
    - using exceptions

PrintWriter



+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+PrintWriter(filename: String, csn:String)	Creates a PrintWriter for the specified file and charset
+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

PrintWriter::close()

- Any system resources associated with a PrintWriter should be released
- Use the PrintWriter::close() method
- Why it is important to do “close()” and do it properly?

Write Text to File: First Try

- Observe WriteText.java
- Is there any problem?

```
PrintWriter output = new PrintWriter(file, "UTF-8");
// Write formatted output to the file
output.print("John T Smith "); output.println(90);
output.print("Eric K Jones "); output.println(85);
// doing something more …
// 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, "UTF-8");
// Write formatted output to the file
output.print("John T Smith "); output.println(90);
output.print("Eric K Jones "); output.println(85);
// doing something more …
// 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 the improved WriteText.java

```
PrintWriter output = null;
try {
        output = new PrintWriter(file, "UTF-8");
        // Write formatted output to the file
        output.print("John T Smith "); output.println(90);
        output.print("Eric K Jones "); output.println(85);
} finally {
        // Close the file
        output.close();
}
```

Autoclose using try-with-resources

- JDK 7 provides the followings 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, "UTF-8")) {
        // Write formatted output to the file
        output.print("John T Smith ");
        output.println(90);
        output.print("Eric K Jones ");
        output.println(85);
}
```

Questions?

- Writing text using File and PrintWriter
  - What are the approaches to release system resources used by PrintWriter?
    - Two patterns
      - The finally block
      - Try-with-resources

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.
+Scanner(File source, String csn)	Creates a Scanner object to read data from the specified file.
+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):	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, "UTF-8");
    PrintWriter output = new PrintWriter(targetFile, "UTF-8");) {
  while (input.hasNext()) {
    String s1 = input.nextLine();
    String s2 = s1.replaceAll(args[2], args[3]);
    output.println(s2);
  }
}
```

- Change it to use the try-finally pattern?

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.
  - 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.
  - Make sure that you catch as the most specific exception as you can.

Exercise 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 ReplaceText john filename.txt

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

- 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.
- Make sure that you catch as the most specific exception as you can.