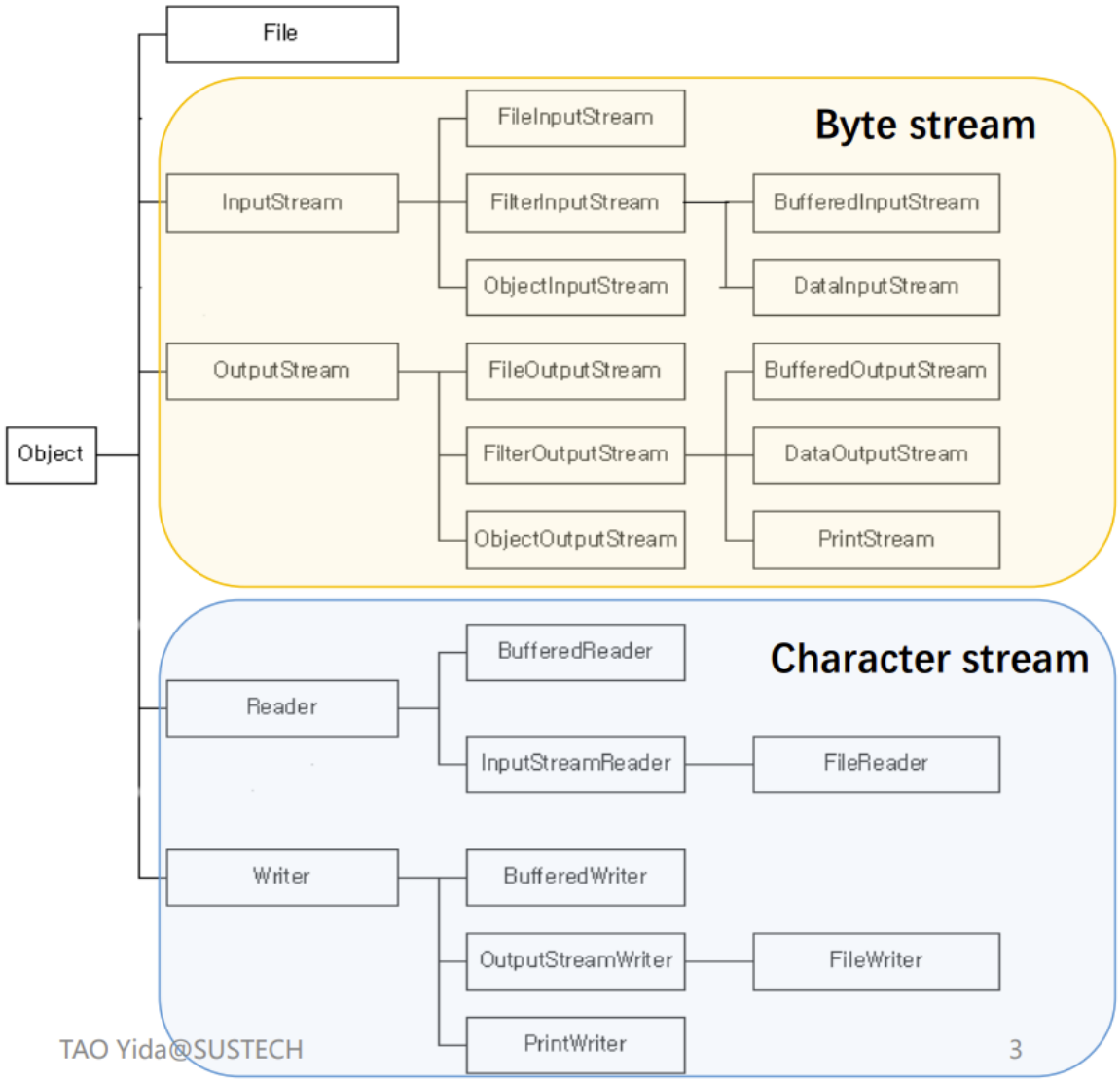
Lecture 5 & Lab 5

I/O Overview and encoding

1. overwivew

   in `java.io` package

   hierachy:



2. Encoding

   convert characters to other formats, often numbers, in order to store and transmit them more effectively.

   1. ASCII

      • represent text in computers

      • using 7 bits to represent 128 chars

      • extended ASCII uses 8 bits for 256 characters

   2. GB2312, GBK, GB18030

• GB stands for 国标

• GB2312 uses 2 bytes(cover 99% usages)

• GBK extends GB2312

• GB18030 extends GBK

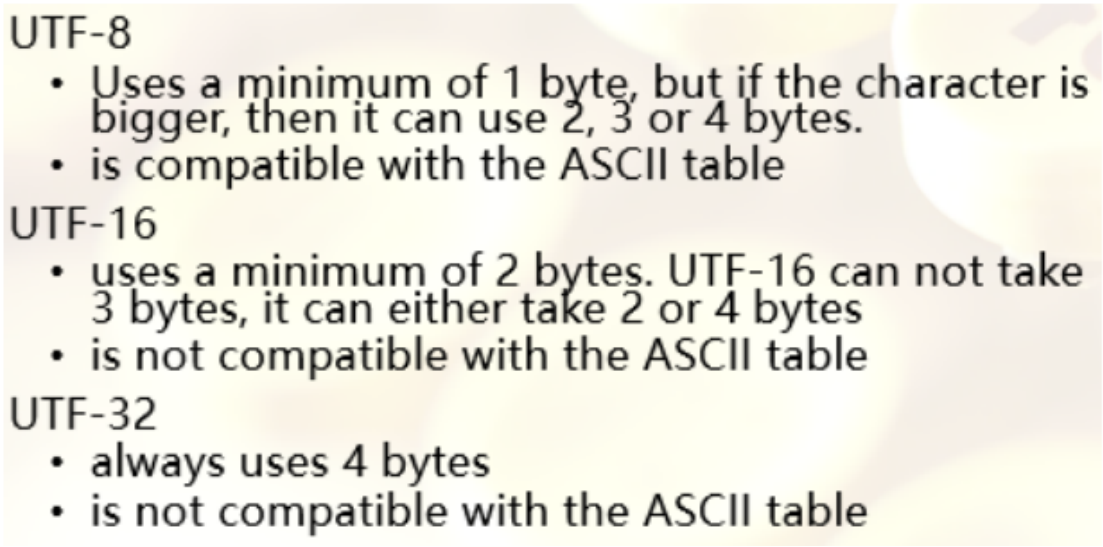
3. problems

   different contries with different language systems implement their own character encoding

4. Unicode

   • movitate by the need to encode characters in all languages consistently without conflicts

   • Unicode is a standard (defines the mapping to code point)



5. utf-8

   • UTF-8 stands for “Unicode Transformation Format – 8-bit”

   • Characters are encoded with varied lengths (1~4 bytes)

Character Range	Encoding
0...7F	$0a_6a_5a_4a_3a_2a_1a_0$
80...7FF	$110a_{10}a_9a_8a_7a_6\ 10a_5a_4a_3a_2a_1a_0$
800...FFFF	$1110a_{15}a_{14}a_{13}a_{12}\ 10a_{11}a_{10}a_9a_8a_7a_6\ 10a_5a_4a_3a_2a_1a_0$
10000...10FFFF	$11110a_{20}a_{19}a_{18}\ 10a_{17}a_{16}a_{15}a_{14}a_{13}a_{12}\ 10a_{11}a_{10}a_9a_8a_7a_6\ 10a_5a_4a_3a_2a_1a_0$

6. Java char implementation

   • 16-bit unsigned int (U+0000~U+FFFF), corresponding to Unicode code points

   • Conversion between int and char refers to the Unicode mapping

   • Characters whose code points are **greater than U+FFFF** are called **supplementary** characters

• Supplementary characters are represented **as a pair of char values** (16 + 16 = 32 bits / 4 bytes)

```
int v1 = 0x0454; // Hex
System.out.printf("%c\n", v1); //є
System.out.printf("%c\n", (char)v1); //є

int v2 = 1108; // Decimal
System.out.printf("%c\n", v2); //є
System.out.printf("%c\n", (char)v2); //є

int v3 = 0x10454; // Hex
System.out.printf("%c\n", v3); //𐑔
System.out.printf("%c\n", (char)v3); //є
```

```
int v1 = 0x0454;
int v3 = 0x10454;
char[] c1 = Character.toChars(v1); // length 1
char[] c2 = Character.toChars(v3); // length 2
System.out.println(c1); //є
System.out.println(c2); //𐑔
```

Byte streams & Charater Streams

1. difference

   Byte Stream

   • for processing **bytes** file, such as picture, music, or large file.

   • Byte streams are **inconvenient for processing info stored** in Unicode

   • Used to

   Character Stream

   • for processing **characters** file, such as `.txt`, `.csv`, `.html`...

   • These classes have read and write operations that are based on **char** values rather than byte **values**

2. Similarity

   • Byte Stream: `InputStream` & `OutputStream` are abstract classes

   Character stream: `Reader` & `Writer` are abstract classes

   • Byte Stream: Subclasses are all called “xxxStream”

   Character stream: “xxxReader” & “xxxWriter”

   • Subclasses for `InputStream` or `Reader` must implement `read()`

• Subclasses for `OutputStream` or `Writer` must implement `write()`

3. `FileInputStream`: Used for reading streams of raw **bytes**
    - `fileInputStream.read()` will read a bytes a time even if text encoding in **utf-8**
    - Same file with different encoding will generate different outcome in the reading.
    - `System.out.println((char)input.read());` will not give a right answer for **utf-8** encoding file
    - useful methods:

```
// Constructor
FileInputStream(String name);
// Open the file named by the path name name in the file system.

// read from file by bytes
public int read() throws IOException;
// Return the next byte of data, or -1 if the end of the file is reached
public int read(byte[] b) throws IOException;
// Reads up to b.length bytes of data from this input stream into an array
of bytes.
public long skip(long n) throws IOException;
// Skips over and discards n bytes of data from the input stream.

// close the file
public void close() throws IOException;
```

4. `FileReader`: Used for reading streams of **characters** (instead of streams of **bytes**)
    1. Java system/platform default encoding

        • Differs from **OS** and **language settings** (e.g., **GBK** on 中文操作系统)

        • Could be changed (environment variable, IDE, code)

    2. File encoding

        • Independent from Java

        • Could be changed

        • When using Java to read a file, the Java system default encoding and the file encoding should be **consistent**

    3. Set a right encoding in `FileReader`

```
Reader r = new FileReader("src/test.txt", Charset.forName("gb2312"))
```

    4. `InputStreamReader` :
        - superclass of `FileReader`
        - use to transfer `InputStream` to `reader`

```
// Constuctor of InputStreamReader
InputStreamReader(InputStream in, String charsetName);

Reader reader = new InputStreamReader(new
FileInputStream("src/test.txt"), "UTF-8")
```

5. Useful methods in `FileReader`: almost the same

```
// Constuctor
// using the platform's default charset
public FileReader(String fileName) throws FileNotFoundException;
// use the specified charset
public FileReader(String fileName, Charset charset) throws IOException;

// method extends from InputStreamReader
public int read() throws IOException;
public int read(char[] cbuf, int off, int len) throws IOException;

// close
public void close() throws IOException;
```

5. Similarly, methods in `FileOutputStream` and `FileWriter`

- `FileOutputStream`

```
// constuctor
//  If append is true, then bytes will be written to the end of the
file
public FileOutputStream(String name, boolean append) throws
FileNotFoundException;

// write()
public void write(int b) throws IOException;
public void write(byte[] b, int off, int len) throws IOException;

// close()
public void close() throws IOException;
```

- `FileWriter`

```java
// constructor
FileWriter(File file, Charset charset, boolean append);

// write
// a character takes at most 4 bytes
public void write(int c) throws IOException;
public void write(char[] cbuf, int off, int len) throws IOException;
public void write(String str, int off, int len) throws IOException;

// close()
public void close()throws IOException;
```

6. Refer to more details

> FileInputStream (Java SE 17 & JDK 17) (oracle.com)
>
> FileReader (Java SE 17 & JDK 17) (oracle.com)
>
> FileOutputStream (Java SE 17 & JDK 17) (oracle.com)
>
> FileWriter (Java SE 17 & JDK 17) (oracle.com)

FilterInputStream

FilterInputStream allows us to **wrap an existing input stream** by extending the InputStream class, and **filter** or **modify** it when reading data.

This approach allows us to perform some **transformations** or **manipulations** on the data as we read it, such as **decrypting or decompressing the data.**

1. FilterInputStream Example I

```java
InputStream zfile = new GZIPInputStream( // + gzip functionality
    new BufferedInputStream( // + buffered functionality
        new FileInputStream(
            "src/test.zip"
        )
    )
)
```

2. FilterInputStream Example II

```java
PushbackInputStream pbin =
new PushbackInputStream( // + pushback functionality
    new BufferedInputStream( // + buffered functionality
        new FileInputStream("test.txt")));

int b = pbin.read();
pbin.unread(b); // added functionality to push back bytes
```

3. FilterInputStream Example III

```
DataInputStream din =
new DataInputStream( // + read-numbers functionality
    new ZipInputStream( // + zip functionality
        new FileInputStream("test.zip")));

// added abilities to read numbers, not only bytes
din.readDouble();
din.readInt();
```

4. direct known subclasses
    - `BufferedInputStream`

      Buffer the input in an array, enhance the speed of reading
    - `DataInputStream`

      A data input stream lets an application read **primitive Java data types** from an underlying input stream in a machine-independent way.
    - `DigestInputStream`

      record the digest message(like a summary)
    - `InflaterInputStream`

      decompre data in `.zip`, `.gzip`, `.zlib`
    - `LineNumberInputStream`

      `read()` line by line
    - `PushbackInputStream`

      Add functionality that push back something has just read.
    - etc.....

Reading/Writing Text Input/Output

1. When working with I/O, we often work with **human-readable** text rather than binary data
2. Java provide two APIs to assist working with text I/O
    - **Scanning**: useful for breaking down formatted input into **tokens** and translating individual tokens according to their data type (`Scanner`)
    - **Formatting**: assembles data into nicely formatted, **humanreadable** form (`PrintWriter`)
3. Using Scanner for reading text files

```
Scanner in = new Scanner(new File("input.txt"));
// read the text file in the specified size
while(in.hasNextDouble()){
    double value = in.nextDouble();
    // process the value
}
```

4. Using PrintWriter for writing text files

```
PrintWriter out = new PrintWriter("output.txt");
out.println("hello, world");
out.printf("total: %8.2f\n", total);
```

The `PrintWriter` class is an enhancement of the `PrintStream` class that you already know—`System.out` is a `PrintStream` object. You can use the familiar `print`, `println`, and `printf` methods with any `PrintWriter` object:

5. Constructing a `Scanner` and `PrintWriter`

```
PrintWriter(String file_name);
Scanner(File file); // remember that it's not a filename
Scanner(InputStream source);
```

I/O from the Command Line

Standard Streams

- Standard streams read input from the keyboard and write output to the display.
- Java platform supports three Standard Streams
- `System` is a `final` class with **private constructor**

Fields

Modifier and Type	Field and Description
`static PrintStream`	`err` The "standard" error output stream.
`static InputStream`	`in` The "standard" input stream.
`static PrintStream`	`out` The "standard" output stream.

`System.err`

`System.in`

`System.out`

`System.in`

- Standard input, often read keyboard input
- To use **Standard Input** as a **character stream**, wrap `System.in` in `InputStreamReader`.
- `System.in` with `Scanner`: Parse the input into different primitive types and strings

```
Scanner in= new Scanner(System.in);
```

`System.out`

- `System.out` is defined as a `PrintStream` object
- Although it is technically a byte stream, `PrintStream` **utilizes an internal character stream object to emulate many of the features of character streams** (same for `PrintWriter`)
- Could use `setOut()` to **redirect** the output to other resources

```
// construct a new PrintStream with a specified file
PrintStream out = new PrintStream(new File("src/sysout.txt"));
// re-assign the standard output from console to file
System.setOut(out);
// this will be written to file
System.out.println("where am I?");
```

notes: out is a final field, for a specialization, compiler will change it if we use `setOut()`

- `System.out.println`: All things will be printed with no filter.