I/O in Java

Reading and Writing Data

Review

What is the point of enum?

a) It lets you create named constants

b) It lets you associate named constants with values, like a description

c) It makes switch/case statements more readable

d) All of these

Review

Which string does the regex r'day|night' match?

a) night and day

b) day and night

c) night or day

d) It matches all of these

Objectives

- Topics
  - Basic Java I/O classes
  - Basic file reading and writing
- Goals: after this lecture, you will be able to
  - describe the basic i/o classes
  - use these classes to read and write files

Java I/O Classes

- The Java I/O class hierarchy is large and confusing
  - Bytes, characters, streams, data, …
  - Many class names sound the same
- If you know a few basic patterns, the other stuff can be looked up if you need them
- Without using some complicated regex parsing, you need to know the type and format of the data you're reading or writing
  - If you're writing data that will be read in later, you're in charge
  - Not everything is CSV

Basic I/O Hierarchy

- ***Stream:*** fancy name for an input source or output destination; includes files, keyboard/screen, network devices
- ***Byte streams***: read/write single bytes
  - InputStream, OutputStream hierarchies
- ***Character streams***: read/write Unicode characters of some length
  - Reader, Writer hierarchies
  - These were added for internationalization and speed
- ***Random access***: read/write binary records
  - Stands alone from the other I/O classes

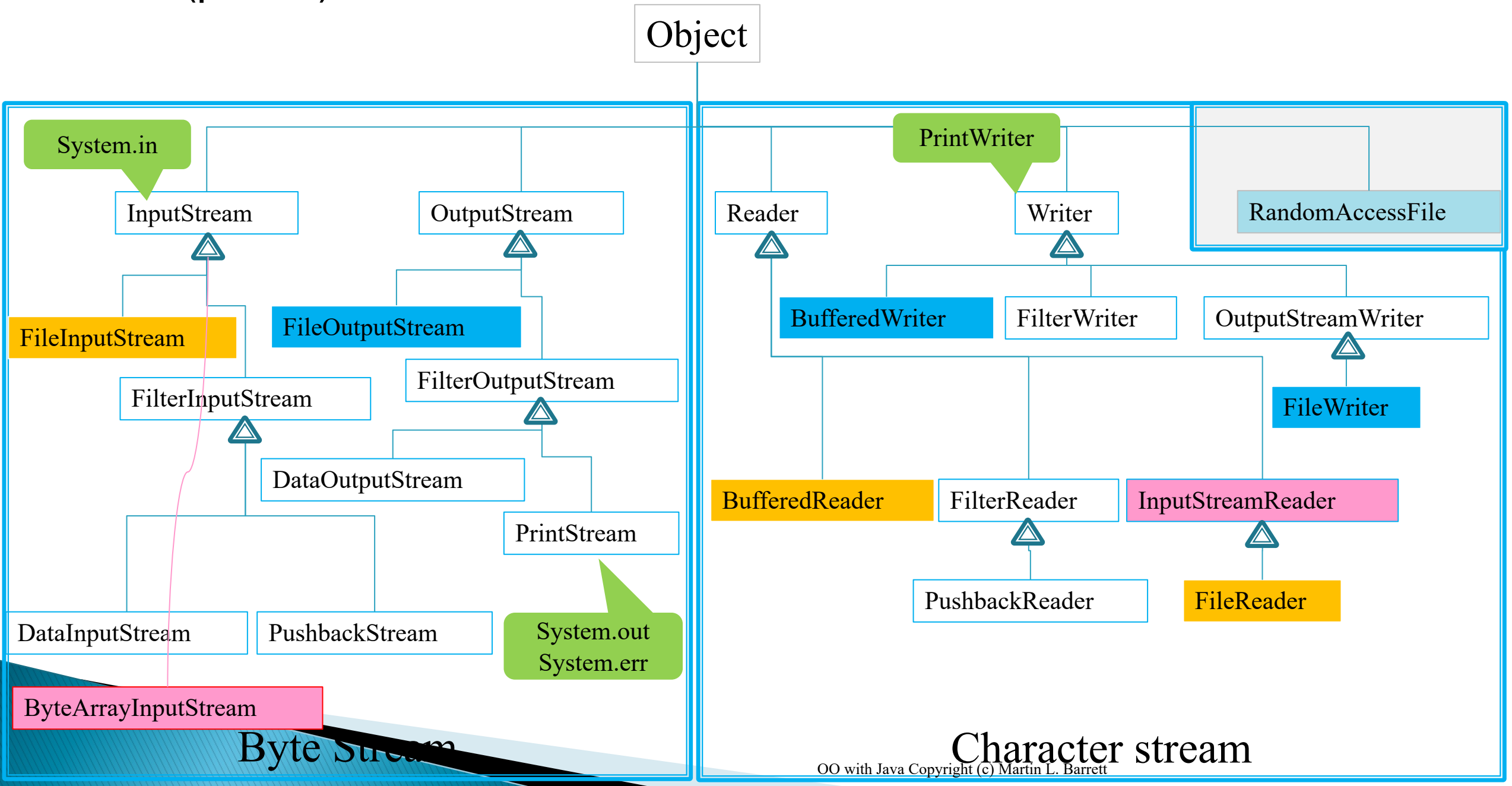
Buffered versus Unbuffered

- ***Buffered*** means the JVM reads more data from a file than it currently needs, so that the next read(s) will be faster – it's already in memory
- ***Unbuffered***: means each read or write request is handled directly by the operating system
  - So much slower

Well?

- If you have the choice, use buffered character streams
  - faster, UTF-oriented
- And wrap them inside some useful class to make your life easier
  - Scanner for input
  - PrintWriter for output
- But some libraries require that you use some other classes, so check the docs

Java.io (partial)



File I/O

- ***FileReader*** and ***FileWriter*** are unbuffered – for every read/write, the JVM contacts the OS, slowing things down – and read character by character
- ***BufferedReader*** and ***BufferedWriter*** use buffering: local storage of previously read/written data to speed things up
- Typically, you wrap the former inside the latter:

```
BufferedReader input = new BufferedReader( new FileReader(filename));
```

- BufferedWriter and FileWriter work similarly

File I/O, cont.

```
public static String read( String filename ) {
    try ( BufferedReader input = new BufferedReader(
            new FileReader(filename)) ){
      String s; StringBuilder sb = new StringBuilder();
      while ( (s = input.readLine()) != null ) {
                sb.append(s+"\n");
      }
    } catch (IOException e) { System.out.println(e.getMessage()); }
    return sb.toString();
}
```

try with resources

readLine() returns null at eof

Scanner

- The ***Scanner*** class is not part of the I/O library (it's in java.util), but it can be used to wrap other classes for parsing out data from a buffered stream
- We've seen this before:

```
Scanner keyboard = new Scanner( System.in );// Wraps an InputStream
```

  - System.in (Standard input) is normally the keyboard, but it can be redirected from a file or command output at the OS level
- But it has constructors to wrap File, Path, and any Readable (FileReader, BufferedReader)

Scanner, cont.

```
// This reads a file of int data into an ArrayList
public static void readInts( String filename )  {
    try ( Scanner scanner = new Scanner(
            new BufferedReader(
                new FileReader(filename))) ) {
        while ( scanner.hasNext() ) {
            intList.add(scanner.nextInt());
        }
    } catch (IOException e) { System.out.println(e.getMessage()); }
 }
```

use hasNext() or hasNextInt()

ArrayList<Integer>
declared elsewhere

PrintWriter

- The ***PrintWriter*** class is a child of Writer
- But it has constructors to wrap File, OutputStream, or Writer
- Various overloads of print( ) and println( ), plus format( )

PrintWriter, cont.

- System.out and System.err are PrintStream objects, but basically have the same formatting capabilities
  - System.out (Standard Output) is normally the screen but can be redirected at the OS level
  - System.err (Standard Error)is also normally the screen, can also be redirected, and is used for things like traceback stacks. Some apps redirect this to a file so that the user doesn't see it (normally), but it's there for developers to help debug

PrintWriter, cont.

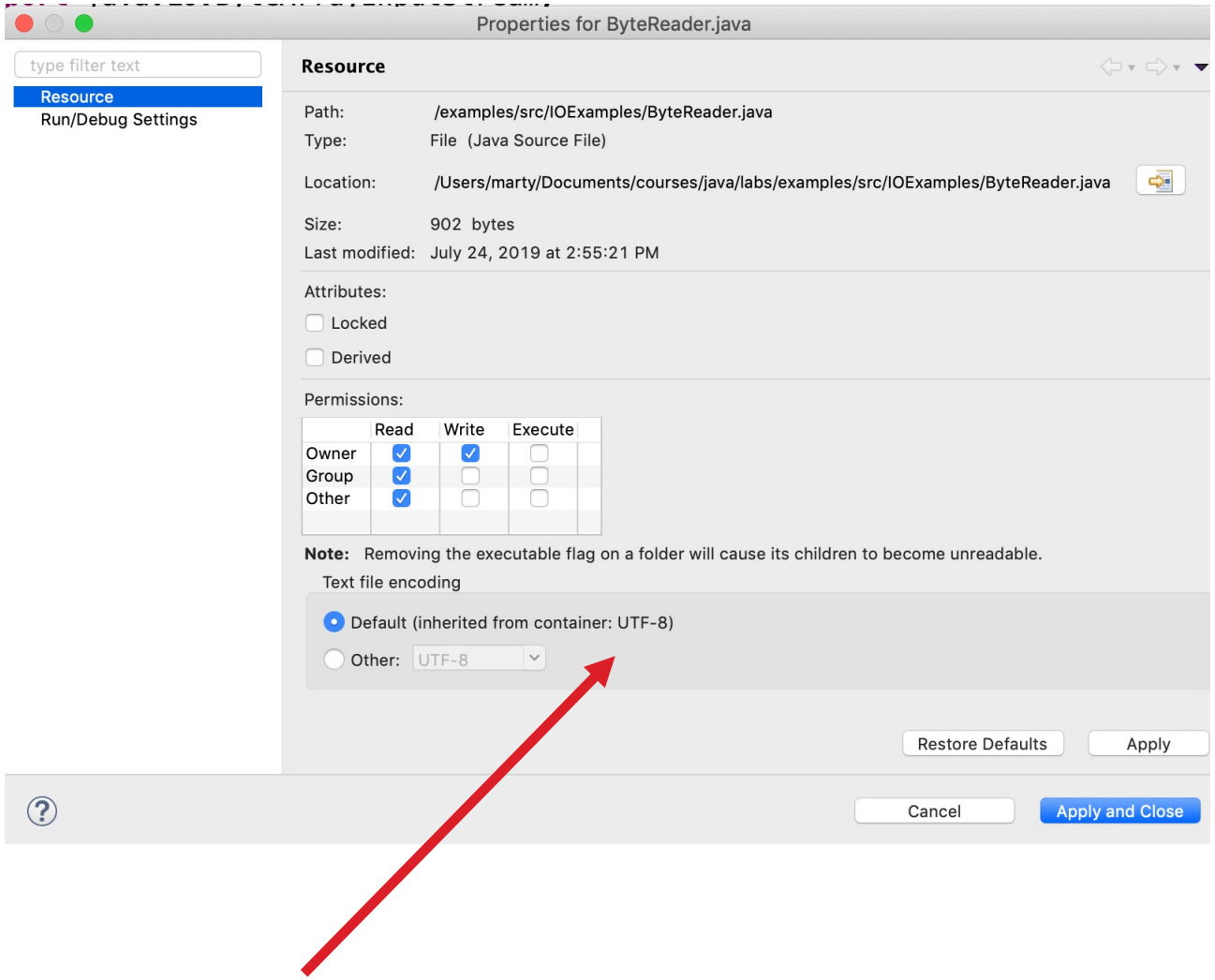
```
public static void writeInts( String filename ) {
    try ( PrintWriter printWriter = new PrintWriter(
            new BufferedWriter(new FileWriter(filename))) ) {

      for (Integer i: intList) {
          printWriter.println(i);
      }
    } catch(IOException e) {
        System.err.println(e.getMessage());
    }
}
```

Write to the error channel

Other I/O Classes

- The next example uses an InputStreamReader wrapping a ByteArrayInputStream to interpret some UTF-8 characters
- You can see what encoding is the default by looking at the Properties->Resource page for a project or data file



Other I/O Classes, cont.

- You can always write raw UTF characters using their /u codes
  - \u0041 : $0*16^3 + 0*16^2 + 4*16^1 + 1*16^0 = 65$, so A in UTF/ascii
  - \u0905 : $0*16^3 + 9*16^2 + 0*16^1 + 5*16^0 = 2309$

\u0905 or 2309 decimal

Devanagari[1]

Official Unicode Consortium code chart (PDF)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+090x	ँ	ं	ः	◌:	अॅ	अ	आ	इ	ई	उ	ऊ	ऋ	ऌ	ऍ	ऎ	ए

https://en.wikipedia.org/wiki/Devanagari_(Unicode_block)

- \u263A: $2*16^3 + 6*16^2 + 3*16^1 + 10*16^0 = 9786$

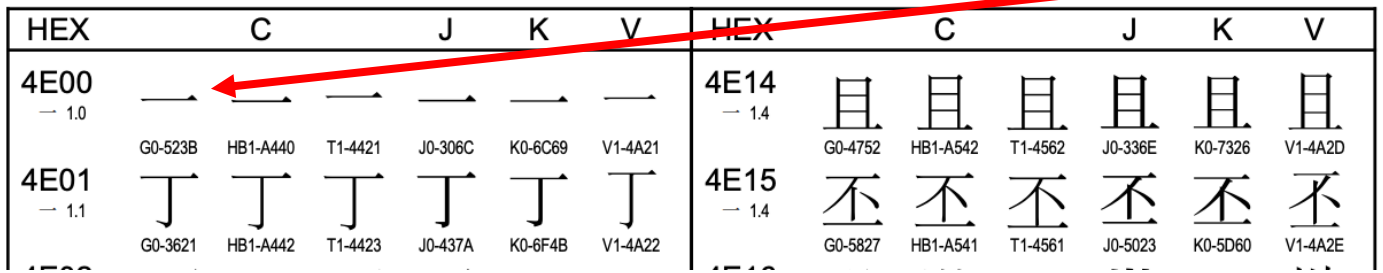
\u263A or 9786 decimal

https://en.wikipedia.org/wiki/Miscellaneous_Symbols

- \u4E00 : $4*16^3 + 114*16^2 + 0*16^1 + 0*16^0 = 19968$

4E00 **CJK Unified Ideographs** **4E26**

\u4E00 or 19968 decimal



https://www.unicode.org/charts/PDF/U4E00.pdf

```java
public class ByteReader {     // This example by Professor Dwivedi

public static void main(String[] args) {
String s = "\u0041\u0042\u0043\u0044\n"          //Latin
          + "\u0905\u0906\u0907\u0908\n"          //Devnagari
          + "\u266A\u266B\n"                      //Musical notes
          + "\u263A\n"                            //Smiley
          + "\u0627\u0628\u0629\u062A\n"          //Arabic
          + "\u4E00\u4E8C\u4E09\u4E96";           //CJK
```

ABCD
अआइई
♪♫
☺
ابـة ت
一二三亖

```java
byte[] codes = s.getBytes(); //load s as raw bytes

// Open InputStreamReader with a character decoder as per UTF-8
try (InputStreamReader isr = new InputStreamReader(new ByteArrayInputStream(codes),
          StandardCharsets.UTF_8);) {
          int c;
          while ((c = isr.read()) != -1)
                    System.out.print((char)c); //will print according to settings in IDE
} catch (IOException e1) {
          e1.printStackTrace();
}
} }
```

Other I/O Classes, cont.

- The DataInputStream and DataOutputStream classes support I/O of binary primitive data
- They have methods readInt(), writeInt(), readDouble(), writeDouble(), and so on
- readUTF( ), writeUTF() read/write a UTF String
- For some reason, readLine( ) has been deprecated